Vertex Cover

Algorithms

Cover algorithm (algo_couplage in the code)
Input : graph G=(V,E)
Output : a cover of G
Algorithm : At first, we have an empty set C. Then we look at each edges of G and if none of the two ends of the edge are in C, we add them in C. We return C.
This algorithm always returns a cover but not necessarily with the minimum number of vertices. It has an approximation factor of 2.

Glutton algorithm (algo_glouton in the code)
Input : graph G=(V,E)
Output : a cover of G
Algorithm : At first, we have an empty set C. While E is not empty, we add the vertew of maximum degree in C and remove its edges from E. We return C.

Branch algorithm (branchement_simple in the code)
For each edge (u,v), we consider two cases : either u is in the cover or v is in the cover. Starting with the initial graph G and an empty set C representing the solution, we create a first branch where u is in the solution and continue the reasoning on the graph without u. Symmetrically, we consider a second branch with v in the solution and where we continue the reasonning on the graph without v.

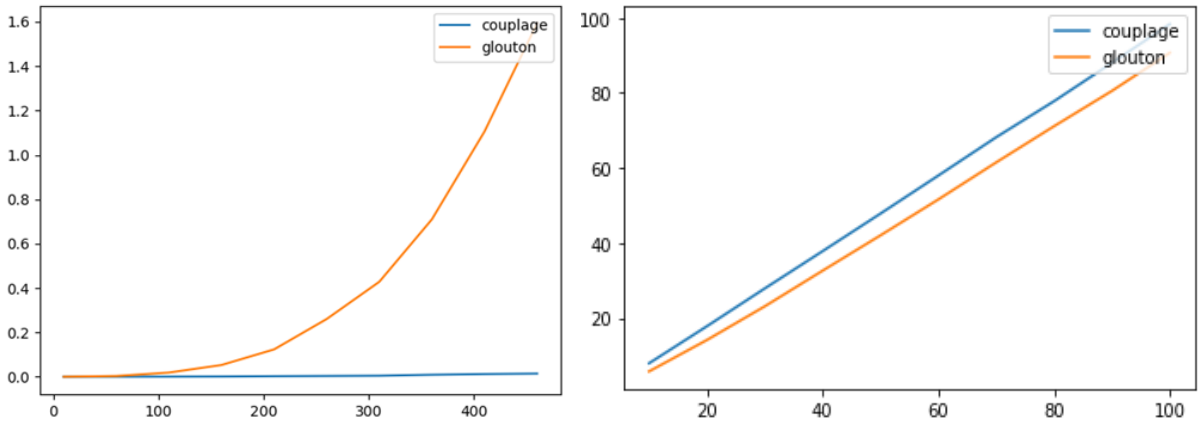
Branch and Bound algorithm
The principle is the same as the branch algorithm but before enumerating the candidate solutions of a branch, the branch is checked against upper and lower estimated bounds on the optimal solution.
We have four different algorithms of type branch and bound :
_ the first one (branchement_borner in the code) is bounded by $(2n-1-\sqrt{(2n-1)^2-8m})/2$ with n the number of vertices of G and m the number of edges of G.

_ the second one (branchement_borner3 in the code) is a version where we separate the edge (u,v) with u in one branch and in the second branch with v, we remove u (since the case where u is in the solution is handled in the first branch). Since u is not in the solution, its neighbourhood is in the solution.

_ the third one (branchement_borner2 in the code) is the same as the second one but when we choose the edge where we create the two new branches, we choose the edge (u,v) with u of maximum degree to eliminate a maximum number of vertices in the second branch.

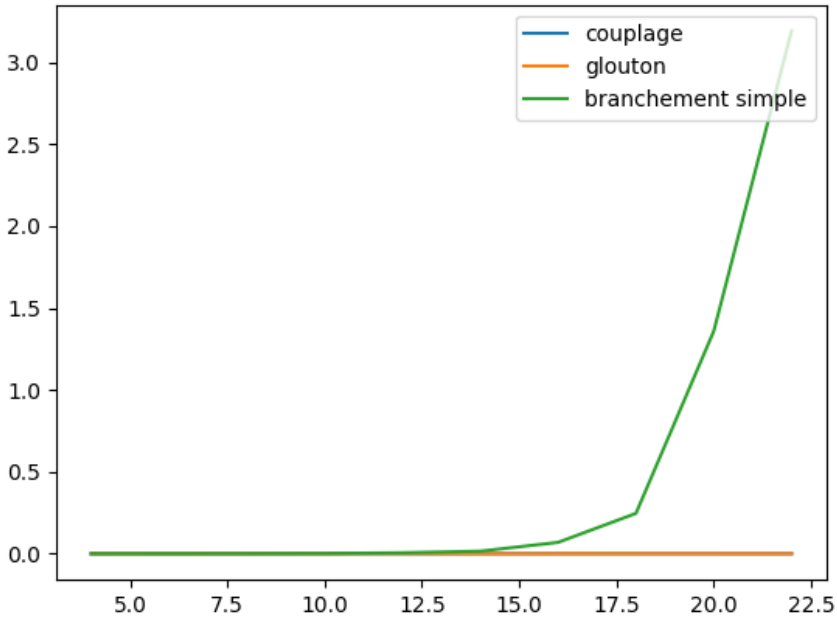
_ the fourth one (branchement_borner5 in the code) is the same as the third one but where we also remove the vertices of degree 1.

Comparison of the different algorithms

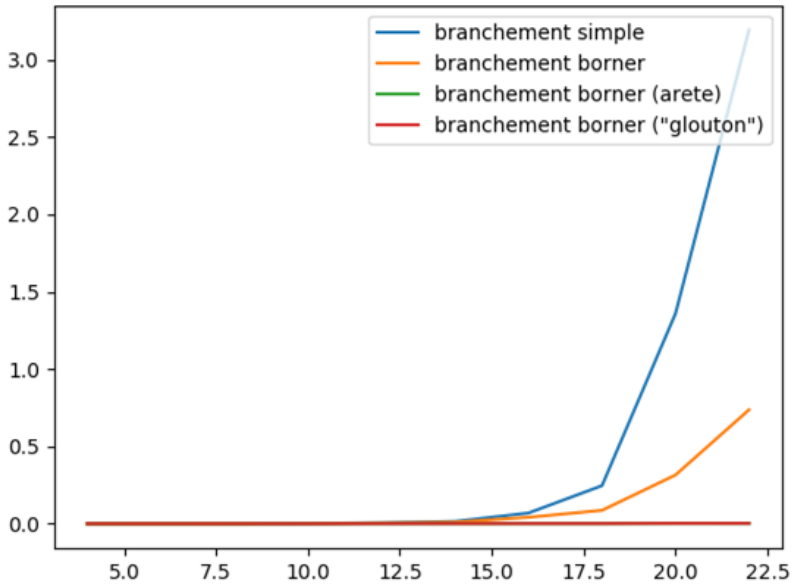


The first graph represents the evolution of time according to the size of the graph (in blue : cover algorithm ; in orange : glutton algorithm).
The second graph represents the evolution of the size of the solution according to the number of vertices in the graph (in blue : cover algorithm ; in orange : glutton algorithm).

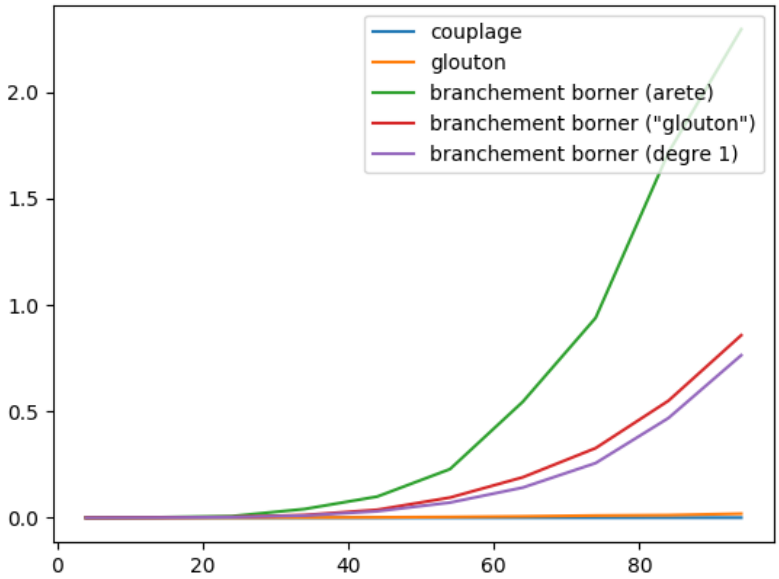
The glutton algorithm returns better solutions in average than the cover algorithm. However, its computation time is higher than the cover algorithm : O(n) for the cover algorithm and $O(n^2)$ for the glutton algorithm.



This graph represents the evolution of time according to the size of the graph (in blue : cover algorithm ; in orange : glutton algorithm ; in green : branch algorithm).
The curve of the cover algorithm is covered by the glutton algorithm. The branch algorithm seem to behave like a complexity of $O(2^n)$.



This graph represents the evolution of time according to the size of the graph (in blue : branch algorithm ; in orange : first branch and bound algorithm ; in green : third branch and bound algorithm ; in red : second branch and bound algorithm).
The curve of the third branch and bound algorithm is covered by the second branch and bound algorithm.
With the addition of bounds, we don't look over all the branches of the tree which naturally reduces the computation time.
The choice of a maximum degree vertex in each nodes of the tree allows even better performances.



This graph represents the evolution of time according to the size of the graph (in blue : cover algorithm ; in orange : glutton algorithm ; in green : third branch and bound algorithm ; in red : second branch and bound algorithm ; in purple : fourth branch and bound algorithm).
Although the detection of one degree vertices has a cost (O(n) in our implementation), the removal of one degree vertices allows better performances (about 20% faster than the second branch and bound algorithm.