NP3: Graph Problems

Notes for CS-8803-GA: Introduction to Graduate Algorithms

Georgia Tech (Dr. Eric Vigoda), Fall 2017

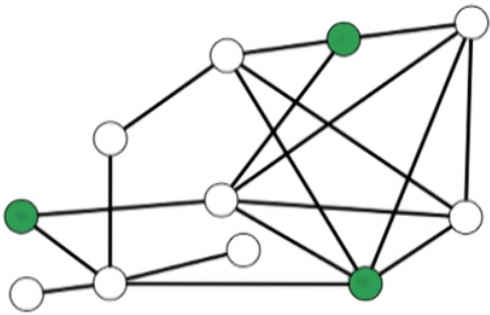
as recorded by Brent Wagenseller

Outline

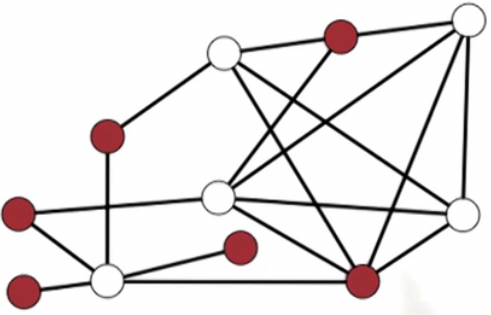
* We have shown that 3SAT is NP-Complete using SAT → 3SAT
  + We assumed SAT was NP-Complete for us to be able to do this
* Now we are going to show the following are NP-Complete:
  + Independent Sets
  + Clique
  + Vertex Cover

Independent Set

* An independent Set is as follows: For an undirected G=(V, E), subset S ⊂ V is an independent set if no edges are contained in S
  + i.e. for all x,y ∈ S, (x, y) ∉ E
  + In other words, the subset of vertices are not directly connected via an edge
* Graphical example (green is subset S)



* Another example:



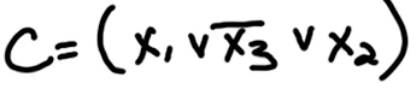
* Examples can be trivial, like the empty set or 1 vertex in the set; however, the interesting problem is to find the largest independent set possible
* The maximum size independent set problem is NOT in NP; this is because we have no easy way to verify if its truly a Max (much like Knapsack)
* The Search Version of Independent Set
  + Much like Knapsack, we can make the Independent Set problem NP by changing its mechanics into a **search version**: instead of finding the max, use a goal instead
    - PLEASE NOTE: The version that tries to find the max (the original Independent Set problem) is known as the **optimization version**, as it is optimizing the solution
  + Input: undirected G=(V, E) and goal g
  + Output: independent set S with size |S| >= g if one exists; NO otherwise
  + Theorem: The independent Set problem is NP-Complete

Proving the Search Version: Independent Set is NP-Complete

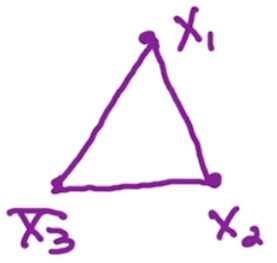
* We have to show: Independent Set ∈ NP
  + NOTE: From this point out, by ‘Independent Set’ I mean ‘Independent Set: Search Version’
  + Given input G, g and solution S, we need to verify S is a solution in polynomial time
  + In O(n2) time we can check all pairs x,y ∈ S: verify (x, y) ∉ E
    - We can check all (x,y) pairs in S by verifying they are not incident
  + In O(n) time we can check |S| >= g
* We have to show: a known NP-Complete Problem can be reduced to the Independent Set problem
  + We will use 3SAT → Independent Set
    - Note: whenever there is a choice, chose 3SAT over SAT; its easier
  + Consider 3SAT
    - Input f with variables x1, …, xn and clauses C1, …, Cn.
    - Each clause has size |Ci| <= 3
  + Since the input to the Independent Set is a graph, we are going to have to construct a graph G from the 3SAT input
    - We ALSO have to set our goal g; this will be the number of clauses m
  + Idea: For each clause Ci, create |Ci| vertices
  + <proof will continue after a more thorough problem explanation>

Further Reduction Setup

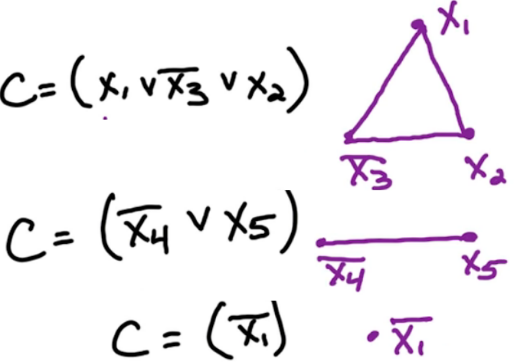
* Clause Edges
  + Consider a clause C:



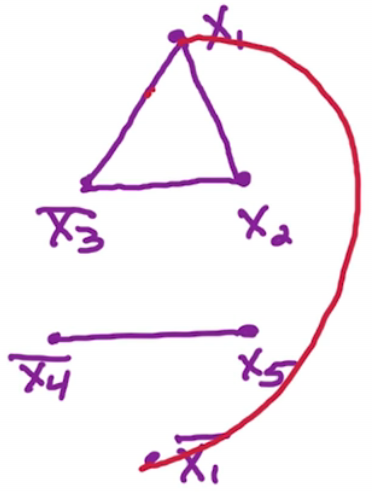
* + Our graph will have 3 corresponding vertices to the literals



* + - Note that other clauses may have vertices for these literals as well
      * its entirely possible that, say, x2 is used in another clause
      * If a literal is used elsewhere, it will NOT use this vertex – it will use its own version (so x2 may appear multiple times as a vertex)
    - We add exactly two edges for each vertex – one edge to each of the other vertices; these are known as **clause edges**
    - An independent set S has <= 1 vertex per clause
      * So this means we will pick one and only one vertex
    - Recall our goal was to find a set of size g (which was equal to m, the number of clauses)
      * Since g=m, solution has =1 vertex per clause (the satisfied literal)
        + There may be other satisfied literals, but for our problem this does not matter
      * This will ensure a correct answer for 3SAT AND the specified Independent Set problem (as there is guaranteed one satisfied literal per clause)
      * The goal of these seems to be to find a NP-Complete problem and try to get it to fit within the parameters of the target problem; even if it is an odd problem (like trying to fit a bunch of disjoint portions of a graph into the IS problem to accommodate the 3SAT problem)
  + Example with 3 clauses



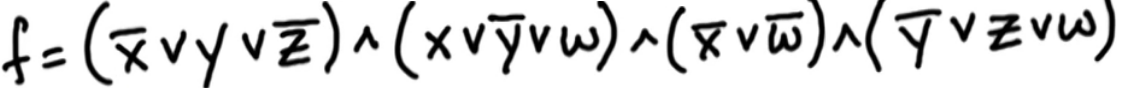
* + - The above is an example of 3 clauses converted to graph problems
    - Notice that every clause is a disjoint graph and has every vertex interconnected
    - Now we would just pick one vertex in each cluster
      * There can be problems with this; say we picked x1, x5, and bar(x1) and satisfied them all (T, T, F)
        + There is obviously not a valid assignment between x1 and bar(x1) as its impossible to satisfy both
        + We must ensure the independent sets correspond to a valid configuration; this is where variable edges comes into play
* Variable edges
  + **Variable edges** are added edges that ensure a variable is never associated with its negative
  + For example, in the above graph we would add an additional edge between all x1 and bar(x1) as follows:



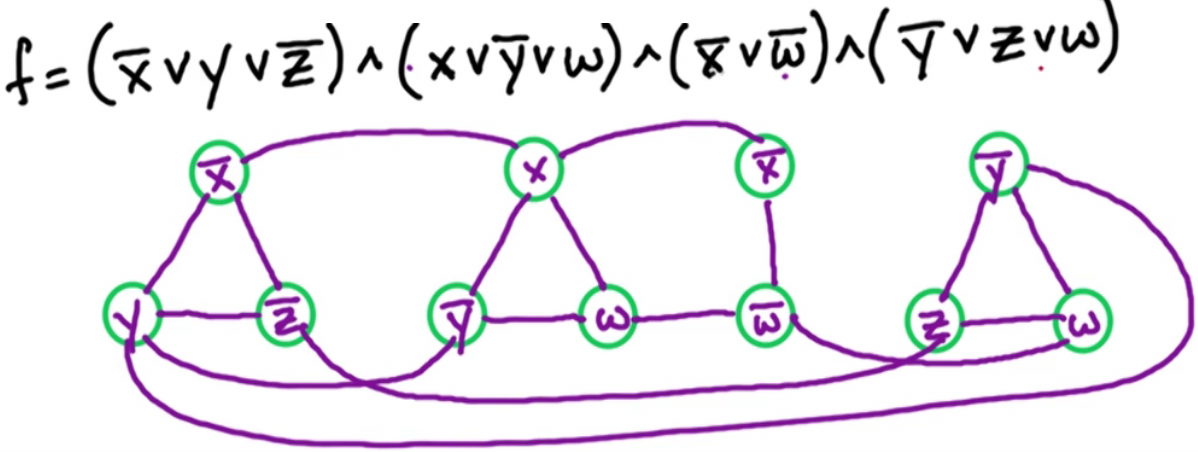
* + For each xi: add edges between all xi and bar(xi)
* Now that clause edges AND variable edges have been added, we can ensure that every clause will be satisfied AND invalid configurations are impossible

Example: 3SAT → IS

* 3SAT setup



* + Variables x, y, w, z
  + 4 clauses
* Graph setup
  + Make a vertex, one for each literal (even if there will be repeats)
  + Add clause edges
  + Add variable edges
    - Connect all variables to their opposites
* Graph:



* Now run the Independent Set algorithm on this
  + As it turns out, it will pick exactly one vertex in each clause and ONLY non-conflicting variables
  + Since Independent Search is using the search version of itself and its using a goal g – AND the goal is equal to m, and we have constructed the algorithm to at most pick one variable in a clause – if its not possible to pick exactly g (or m) variables this will return NO; otherwise it will return the set of variables that will enable 3SAT to work
* As an example, the above could pick bar(x), bar(y), bar(w), and bar(y) to satisfy the 4 clauses
  + Note bar(y) was selected twice – this is OK
  + Note z was never mentioned – this is OK too, there are no constraints for z

Proof: 3SAT → IS

* <Note we already have shown that IS ∈ NP so we skip that here>
* We are going to prove: f has a satisfying assignment ⇔ G has an independent set of size >= g
  + That is to say, (3SAT) f has a satisfying assignment if and only if (IS) G has an independent set of size >= g
  + There is a solution for 3SAT if and only if there is a solution for IS
* Start With Forward (this will be done via proof by construction)
  + Consider a satisfying assignment for f
    - We are assuming f is a satisfying assignment for 3SAT
      * For each clause, at least one literal is satisfied
  + Since we are guaranteed at least one literal to be satisfied in each clause, take exactly one literal from each clause and add as a vertex into the set S
  + We know that S contains exactly one vertex per clause
    - |S| = m = g
  + S has exactly one vertex per clause and does not contain any conflicts (x and bar(x) for example)
    - We are guaranteed no conflicts as we got these from f and f was satisfied and free of error
      * This means there are no variable edges
    - There is 1 vertex per clause because there are no clause edges
  + Since there are no edges, this must mean S is an independent set with size = g = m
  + Same proof, as described by Dr. Vigoda (officially)

(\Rightarrow): Consider a satisfying assignment for f. For each clause C, at least one of the literals in Cis satisfied. Choose one of these satisfied literals and include its corresponding vertex in S. Hence, |S| = m = g.  
By construction, Sincludes exactly one vertex per clause so no edges within a clause’s gadget are covered by S.  Moreover, since we started from an assignment for the variables x_1,\dots,x_nthen Sdoes not include opposing literals, so no edges between opposite literals are covered. Hence, Sis an independent set.

* Now prove the reverse
  + Consider an Independent Set S of size >=g
  + Since we know m = g, we know the set contains 1 vertex per clause
    - Specifically, it responds to a literal set to True
  + Since every clause has a satisfied literal, we know every clause is satisfied
    - There are no contradictory literals since edges were defined as xi to bar(xi)
  + Since this is a valid assignment, it satisfies every clause for 3SAT and therefore satisfies f
  + Same proof, as described by Dr. Vigoda (officially)

(\Leftarrow): Take an independent set Sof size \geq g. For each vertex in Sset the corresponding literal to T, we will prove this gives a satisfying assignment.   Since there are edges between vertices corresponding to opposite literals, Sdoes not contain vertices corresponding to opposite literals and so this assignment from Swill not set both literals x_iand \overline{x_i}to T; hence, it is a valid assignment.  Moreover, since |S|\geq gand since there is a complete graph for each clause’s gadget, the set Shas exactly one vertex per clause’s gadget.  This gives at least one satisfied literal per clause and hence we have a valid assignment that satisfies f.

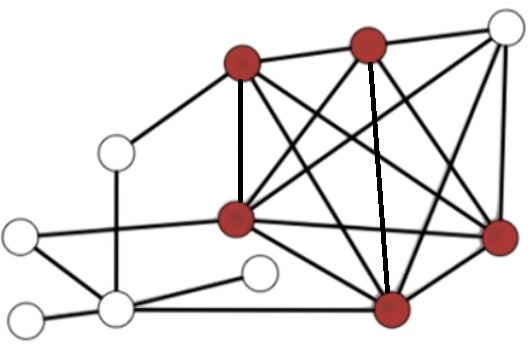
* This completes the proof that the Independent Set problem is NP-Complete

NP-Hard

* Lets re-visit the optimization version of Independent Set
* Note that even though Max-IS (the optimized version) is not in NP, we can still reduce the search version of IS into the optimized version of IS fairly easily: IS → Max-IS
  + For IS, we are looking for at least g, but we are looking for the max in the max set
  + If we find the Max and we check to see if its at least size g, it either gives a solution or a NO
* This means we have a reduction from IS to Max-IS, so this means we can reduce ALL NP-Complete to Max-IS
  + Note this does NOT mean Max-IS is NP-Complete, it simply means its ‘as hard’ as NP-Complete
* This introduces a new class NP-Hard; **NP-Hard** are problems as hard as each other, but may or may not be NP
* NP-Complete problems are the hardest in the set NP; NP-Hard problems are at least as hard as everything in the set

Clique

* A **clique** is a fully connected subgraph
* S ⊂ V is a clique if ∀ x,y ∈ S, (x,y) ∈ E
  + That is to say, for an undirected graph G, S is a subset of V is a clique if for all x,y in S, (x,y) is an edge
    - This seems to be the opposite of Independent Set
* Example:



* + All 5 choose 2 pairs are connected by an edge
* Small cliques are easy to find; the challenge is to find the largest possible clique in a graph
* Search Version
  + Much like IS, we must downsize the Clique problem from the optimization version to a search version in order to have this be NP
  + We can do this again by using a goal (g) instead of maximizing the member set of S
    - Officially, we will be looking for a size |S| >= g
    - If no Clique with |S| >= g exists, output NO; otherwise, output the set S

Proof: Clique is NP-Complete

* First: We have to prove that the Clique problem lies in the class NP
  + Given an input (G, g) and a solution S, we have to show in polynomial time that S is correct
  + To verify the solution works
    - For all x,y ∈ S, check that (x,y) ∈ E
    - At most, this takes O(n2) time to consider each pair and then O(n) to make sure they are connected by an edge
  + To verify that |s|>=g
    - This takes O(n) time
  + We now know Clique ∈ NP
* We will have to show that Clique is NP-Complete
  + Which algorithm do we pick? Its best to pick one similar to Clique; since its a graph problem, lets pick Independent Set → Clique

Clique Idea for Proof

* The Independent Set has NO edges within S, and Clique has ALL edges within S
  + The two are opposites; maybe we can take an edge in one where it is not included in the set in the other
  + For G = (V, E), let = (V, ) where = {(x, y): (x, y) ∉ E}
    - The opposite of G is , the compliment; the edges will be the compliment of the edges in G
  + (x, y) ∈ ⇔ (x,y) ∉ E
    - A pair (x, y) is an edge in if and only if the pair is not an edge in E
* Observation: S is a clique in the compliment if and only if S is an independent set in G

Clique - Proof

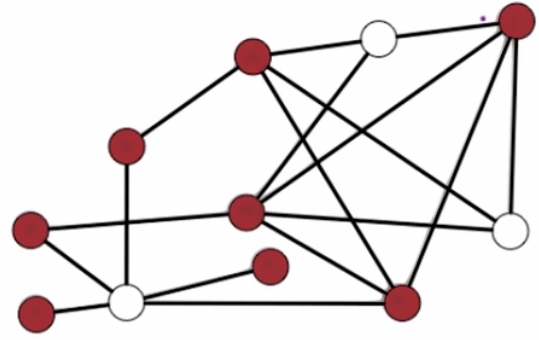
* Observation: S is a clique in the compliment ⇔ S is an independent set in G
* Assumed: We are given input G=(V, E) and goal g for the Independent Set problem
* Let and g be input to the clique problem
  + Recall is the opposite of G
    - That is to say, the edges in are the compliment of G
* If we get a solution S for the clique, then return S for the IS problem; otherwise, return NO
* Therefore, since we can show the Independent Set can be reduced to Clique, Clique is NP-Complete
* Proof, as explained by Dr. Vigoda:

Key idea: Clique is opposite of independent set.  
For a graph G = (V, E), denote its “opposite” graph by: \overline{G} = (V, \overline{E})where \overline{E} = \{(x,y): (x,y) \notin E\}. In other words, (x, y) \notin E \leftrightarrow (x, y) \in \overline{E}.  
Observe that: Sis an independent set in G \leftrightarrow \overline{S} is a clique in \overline{G}.

Now we can show Independent set \rightarrowClique:  
Given G = (V,E) and g as input for the independent set problem, let \overline{G} and g be the input to the clique problem.  
If we get a solution S for Clique then return the same S as the solution to the original independent set problem.  If we get NO, then we return NO for the independent set problem.

Vertex Cover

* Consider a graph; a subset of vertices is a **vertex cover** if it ‘covers every edge of the graph’
  + This means that every edge has at least one of its vertices in the set
* Mathematically:
  + S ⊂ V for every (x, y) ∈ E either x ∈ S and/or y ∈ S
* Example:

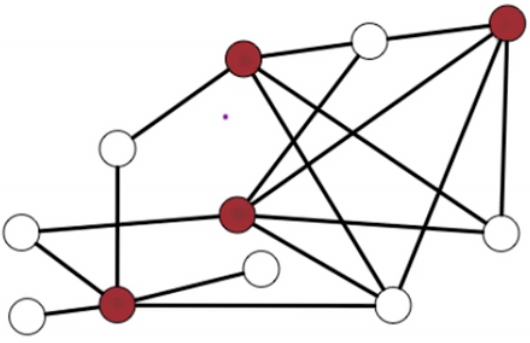


* The optimization problem is to find a minimal set of vertices to satisfy this (as a maximum is easy – select all vertices)
  + Unlike most of the other examples this minimizes the problem
* There is a search version for Vertex Cover
  + Instead of using a ‘goal g’, VC uses a ‘budget b’, as we are finding a min here instead of a max
* Vertex Cover - Search version setup
  + Input: G = (V, E) and budget b
  + Output: Vertex Cover S of size |S|<=b if one exists; NO otherwise

VC: Proof Outline

* We must prove that Vertex Cover is in NP
  + Task: Given input (G, b) and a proposed solution S, verify S is a pertinent solution in polynomial time
  + For every (x, y) ∈ E, >=1 of x or y are in S
    - Check to see if one of the two endpoints is in the set S
    - This takes O(n+m) time
  + Next, check that |S| <= b
    - This can be done in O(n) time
  + We now know that in polynomial time we can check to see if S is a solution; therefore this problem is in NP
* Now we have to show that VC is as hard as everything else in the class NP by reducing a known class to Vertex Cover
  + We will pick Independent Set since it’s a graph, but we could have used Clique as well

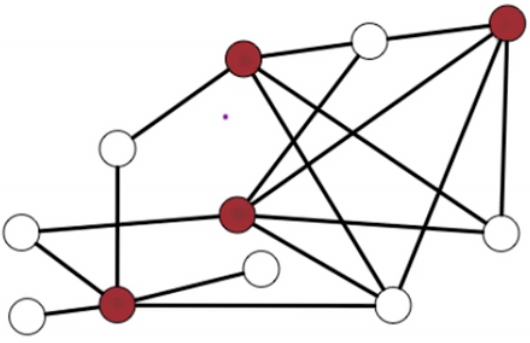
VC: Reduction Idea



* Looking at the example above, its clear that the colored vertices (which is a Vertex Cover set) are the inverse of the non-colored vertices (which happen to be an Independent Set)
* Claim: S is a vertex ⇔ is an Independent Set

Vertex Cover: Proof

* Here is the example where the Vertex Cover vertices are colored:



* Forward: S is a Vertex Cover → is an Independent Set
  + Consider any edge above; at least one endpoint is red
    - In other words, at least one endpoint is in the set S
  + Conversely, every edge has <= 1of x or y in (the compliment of S)
    - At most, every edge has 1 vertex in
    - Due to the nature of minimizing Vertex Cover, taking the opposite vertices yields an Independent Set, as the white vertices will never share an edge with another vertex and will thus have no edge / will be an Independent Set
  + No edge contained in , therefore is an Independent Set.
* Reverse: is an Independent Set → S is a Vertex Cover
  + Take an Independent Set and prove that S is a Vertex Cover
    - We are assuming is an Independent Set
    - No edge is fully contained in
      * At most, only one of x or y or any edge is in s
  + If at most one of (x, y) is in , this means at least one endpoint vertex is in S
    - Thus, every edge of the graph is covered
    - Thus, S is a Vertex Cover
  + The reverse is now proved
* Proof, as laid out by Dr. Vigoda:

(\Rightarrow): Consider a vertex cover S where |S|\leq b. For each (x,y) \in E, x \in S and/or y \in S. Hence, at most one of x, y is in \overline{S}. So no edge has both endpoints in \overline{S}, which means \overline{S}is an independent set.

(\Leftarrow): Take an independent set \overline{S}. For (x,y) \in E, at most one of x,y is in \overline{S}. So at least one of x,yis in S, which means S covers every edge, and hence Sis a vertex cover.

Vertex Cover: Reduction: Independent Set → Vertex Cover

* To convert the Independent Set to get the Vertex Cover input, use the same graph G BUT set b = n – g
  + In other words, b will be the number of vertices minus the goal g of the Independent Set
* If running Vertex Cover on the above can determine a set S where b = n – g, the compliment set will be an Independent Set
  + If it cannot, no Vertex Set nor Independent Set exists with the given parameters
* The steps for reducing the Independent Set to Vertex Cover (and then getting back results that will satisfy the Independent Set)
  + Take the input (G, g) from the Independent Set and create b (b = n-g) and then send (G, b) as input to the Vertex Cover Algorithm
  + Get back set S from Vertex Cover
  + Create the compliment from S and consider as the output from the Independent Set problem

Practice Problems

* Problems
  + 8.4: NP-Completeness Error
  + 8.10: Proof by Generalization
  + 8.14: Clique + Independent Set
  + 8.19: Kite
* Two parts to every problem
  + Show its in NP
  + Take a Known NP-Complete problem and reduce it to the new problem
    - To do this, try to use a similar NP-Complete problem
* Two flavors of NP-Completeness Reductions
  + Proof by generalization
    - Show that the new problem is more general than the used NP-Complete problem
    - Basically, you can set the parameters in the new problem to get the known problem
  + Proof using **gadget**
    - This is what we did for the 3SAT proof
    - Take the formula and modify it some way to change it
    - This has to be done for problems that are not similar on the surface (like a graph vs satisfied formula