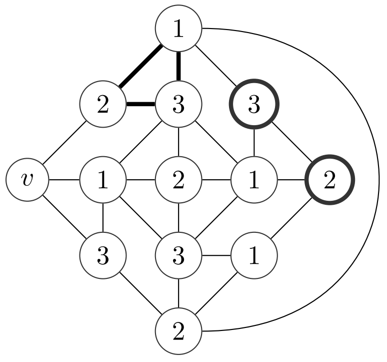
Homework 25

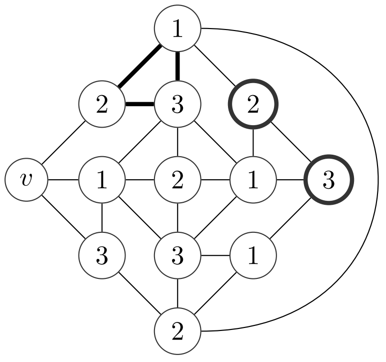
Brian Knotten, Brett Schreiber, Brian Falkenstein

October 26, 2018

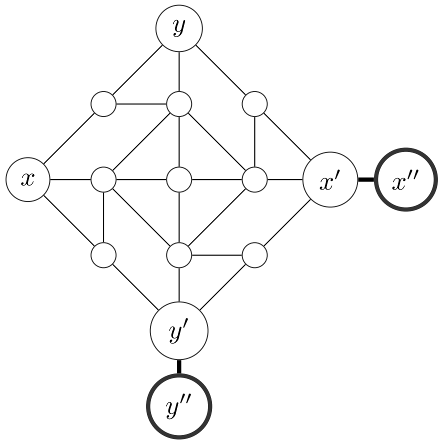
19

In the given crossing gadget G, the vertices y and y' must be the same color for all 3-colorings of G. This is proved via contradiction. Assume y and y' could be colored two different colors in a valid 3-coloring G. Then the graph H, which is identical to G except for an extra edge (y, y') could also be 3-colorable, since y and y' could just be the two different colors. In other words, an extra edge wouldn't affect the colorability if y and y' were able to be two different colors.
First, a triangle within H was arbitrarily chosen to be colored $1, 2, 3$ (denoted by bold edges below). Any other 3-color attempts can be converted into this canonical form by swapping colors within the triangle. For most of the coloring, vertices were forced to be a certain color due to their neighbors. There was only one point where a vertex had a choice of two different colors, which is why we present two graph colorings (differences denoted by bold vertices below). Both canonical graph colorings lead to having a vertex v that is adjacent to all three colors, in which case it can be deemed impossible to 3-color H, contradicting our assumption. Below are the two graph colorings of H:





Since G is symmetrical, it is also the case that x and x' must be colored the same color for all 3-colorings of G. Consider the graph G' identical to G except with two more vertices, x'' and y'', and two more edges, (x', x'') and (y', y''). It must be the case that y' and y'' must be different colors in all 3-colorings of G', since they share an edge, and the same reasoning applies for x' and x''. Furthermore, it must be the case that y and y'' must be different colors, since the color of y is the same as the color of y', which is different from the color of y''. Again, the same reasoning applies to x and x''. Below is G':



Now we can reduce 3-Color to Planar 3-Color. For every crossing in 3-Color's input graph between four vertices, x, x'', y, y'' replace the crossing with the subgraph G' described above. Now there is no longer an edge directly between x and x'' or y and y'', but the property that both pairs must be colored two different colors remains. Moreover, the crossing has been removed. Repeat this process for all crossings to obtain a planar graph P. Run the Planar 3-Color algorithm on P to obtain a 3-Coloring. Color the vertices of the input graph to 3-Color exactly as their corresponding vertices on the output graph have been colored. The transformation from input to input is poly-time as there are at most $|E|$ crossings to replace. The output transformation is also polynomial, as you simply color each of $|V|$ vertices. Therefore if there is a poly-time algorithm to decide if a planar graph is 3-colorable then there is a poly-time algorithm to decide if an arbitrary graph is 3-colorable.

21

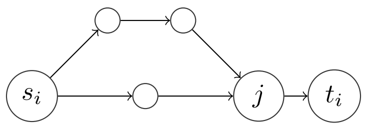
Define the problem described as VDP (vertex disjoint paths). In order to show that $3SAT \geq_{poly} VDP$, we will first show that VDP is self reducible. That is, given the decision problem for VDP, we can determine the paths connecting each (s_i, t_i).

$VDP_{opt} \leq_p VDP_{decision}$

Given as input to VDP_{opt} a graph G and a set of tuples $(s_1, t_1)...(s_n, t_n)$, we can use the decision problem $VDP_{decision}$ to get the actual paths from each s_i to each t_i. Define P_i to be the path connecting s_i to t_i.
First check if $VDP_{decision}$ on G and $(s_1, t_1)...(s_n, t_n)$ returns 1. If not, then paths don't exist. If 1, use breadth first search to get all neighbors of s_1, call these $v_1...v_k$. Now, call $VDP_{decision}$ on G and $(v_i, t_1)...(s_n, t_n)$ for $1 \leq i \leq k$. If this returns 1, add v_i to the path for (s_1, t_1) P_1 and continue this process, searching for neighbors of v_i, etc.
Note that for each vertex added to a path P_i, we must make sure that that vertex does not appear in any other paths. We can enforce this by adding a new tuple to the input. If v_i was the vertex added to the path for some (s_i, t_i), then construct a new (s, t) to be (v_i, v_i). Since the shortest path from v_i to v_i, it will always be an option to have its path be just v_i. This forces other subsequent paths to not include v_i. If we continue this process for all k tuples, we will have the paths for all the inputs.
This transformation is poly time. For each tuple, we must perform BFS to get from s_i to t_i, making one call to $VDP_{decision}$ at each point. Since BFS takes polynomial time in the number of vertices and edges in a graph ($O(|V| + |E|)$, in the worst chase, as all vertices and edges will be visited), and the number of edges is bound by the number of vertices (max E = V^2), if we say the number of vertices is n, than the time for each tuple will be $O(n^2)$. Doing this for all k tuples gives us $(O(kn^2))$, where k can be no larger than $n/2$ (pigeonhole principle, since we cannot repeat vertices, if each vertex in G were assigned to either a source or a sink). Thus, if a polynomial time algorithm exists for $VDP_{decision}$, than one exists for VDP_{opt}.

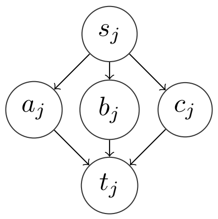
$3SAT \leq_p VDP_{opt}$

Since we have shown that $VDP_{opt} \leq_p VDP_{decision}$, if we have a poly-time algorithm for VDP_{opt}, than we also have a poly-time algorithm for 3SAT (IE, if we assume we have a polynomial time algorithm for decision, than we have one for opt).
We will show how to construct an instance of VDP from an instance of 3SAT, such that the VDP instance is true if and only if the 3SAT instance is satisfiable. To model choosing a variable x_i in 3SAT as either true or false, construct the following sub-graph:



Here, there are 2 possible paths from s_i to t_i, one of length 3 and one of length 4. Due to node j being a bottleneck, only one of these paths can exist in the final output (as j must be a member of both paths, they could not be disjoint). Take the path of length 3 being chosen to mean set x to true, and the path of length 4 being chosen to mean setting x to false (or, setting $\overline{x}$ to true). Construct one such subgraph for each variable in the 3SAT instance.
Now, we must model the clauses in the 3SAT instance. That is, if the 3SAT formula is satisfiable, we will construct a subgraph that asserts that at least one of the literals in that clause will be true. We will construct some component

D_j for each clause $(a_j \vee b_j \vee c_j)$ as follows:



Each clause D_j is considered satisfied if there exists a path between s_j and t_j in the subgraph. So for each clause, add the pair (s_j, t_j) to the input to the VDP algorithm. For a solution to exist, at least one of a_j, b_j, or c_j must be disjoint for use in the path from s_j to t_j. The trick is combining the clauses with the variable choices. Given a variable x_i in the $3SAT$ instance, if x were picked to be true, that means any subgraph corresponding to a clause containing $\neg x$ should not have the path $(s_j, \neg x_j, t_j)$ in the solution set.

In summary: for each variable x_i in $3SAT$, make two paths from s_i to t_i containing the vertices labelled $\neg x_j$ for all j. Add (s_i, t_i) to the input of VDP. Moreover, make three paths from s_j to t_j going through one of a b and c for all clauses with literals $(a \vee b \vee c)$. Add (s_j, t_j) to the input of VDP. Run VDP on this constructed graph and this list of vertex pairs which will return true if and only if the $3SAT$ instance is satisfiable. This algorithm runs in polynomial time, since for each variable only a linear number of vertices and edges are created (depending on how many clauses the variable appears in) and for each clause only a constant number of vertices and edges are created.