Math/CCS 103	Professor: Padraic Bartlett

Minilecture 7: 3SAT and Latin Squares

Week 3 *UCSB 2014*

In today's talk we're going to (mostly) prove the following claim:

Theorem. 3SAT is reducible to the task of completing an arbitrary partial Latin square. In other words, if we have an algorithm that completes arbitrary partial Latin squares, we can turn this algorithm into a method to satisfy boolean formulas in 3SAT.

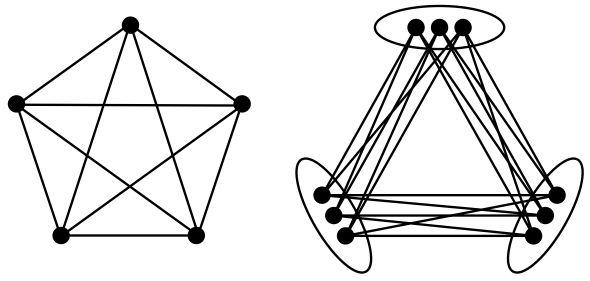
We prove this by starting with the following equivalence:

Theorem. Completing a $n \times n$ partial Latin square is equivalent to the task of triangulating a tripartite graph $G = V_1, V_2, V_3$ whose tripartite complement is triangulatable and whose parts V_1, V_2, V_3 are all size n.

That's... a lot of terms we haven't defined yet in class. Let's break those down here:

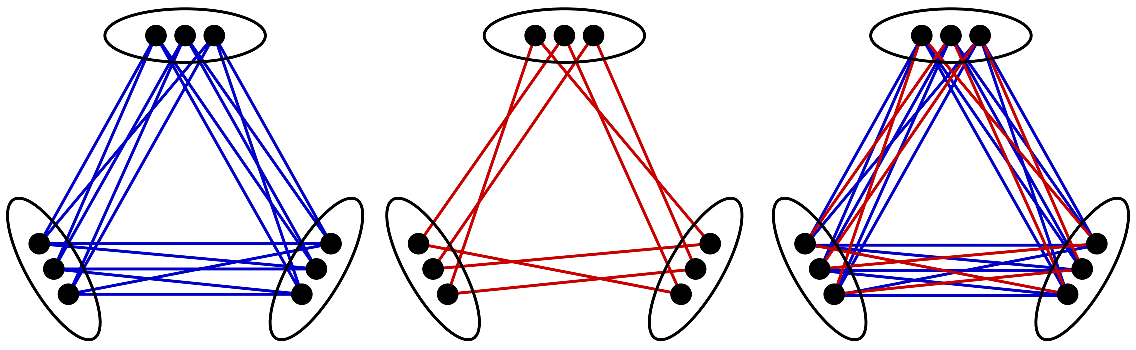
Definition. A graph is called **tripartite** if there is some way to group its vertices into three sets V_1, V_2, V_3 such that no edge in our graph has both of its endpoints in the same V_i.

Here are some examples:


(not tripartite) (tripartite)

Definition. Given a tripartite graph G with tripartition V_1, V_2, V_3, the **tripartite complement** $\overline{G}$ of our original graph is the tripartite graph formed by connecting two vertices $v \in V_i, w \in V_j$ whenever $i \neq j$ and the edge $\{v, w\}$ did not exist in the original graph G.

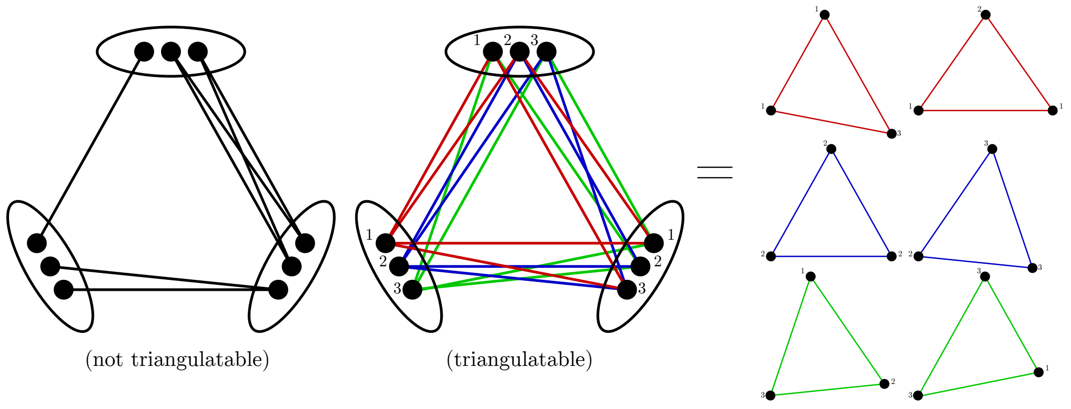
We again give an example:


(original graph) (complement graph) (original superimposed on complement)

Notice that in the picture above, the edges of the complement graph were precisely the edges between distinct parts that did not exist in the original graph.

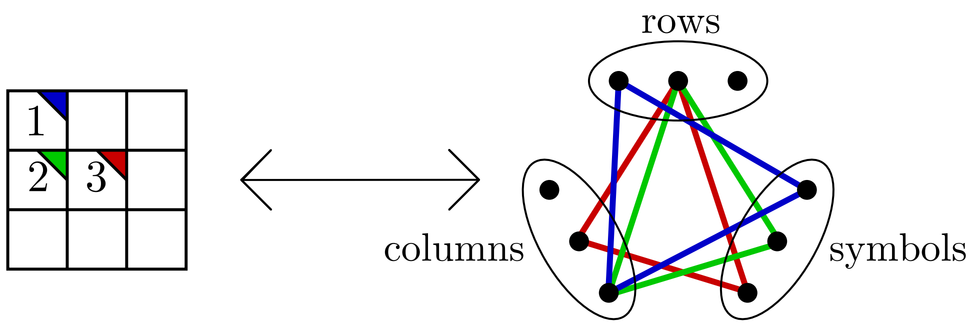
Definition. A **triangulation** of a graph G is a way to divide G's edges up into disjoint subsets, each of which forms a triangle.

We draw two examples below:

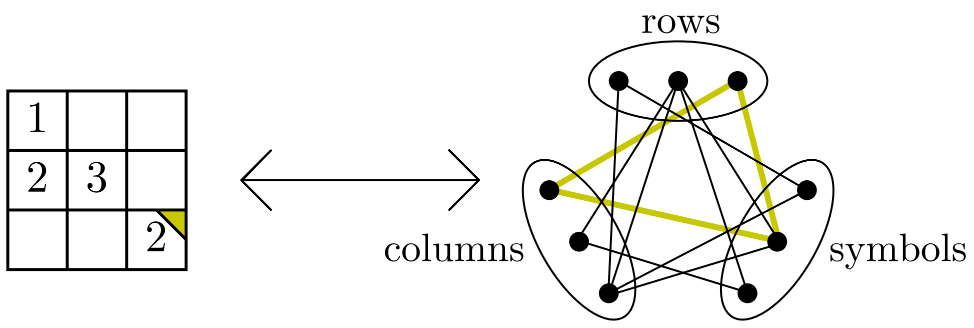


With these definitions, it's actually pretty easy to prove our claim:

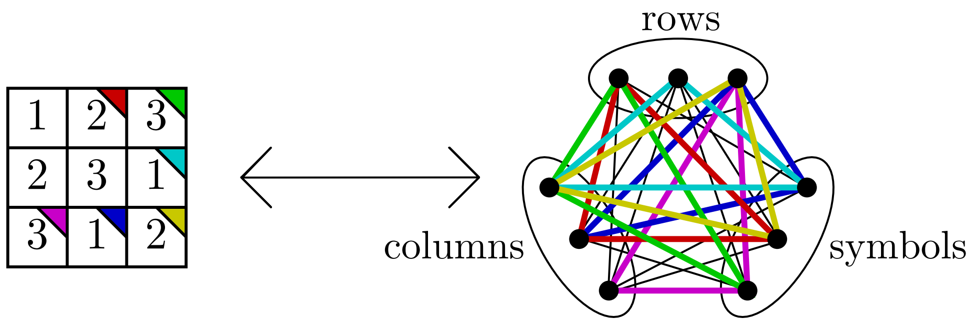
Proof. First, notice that we can turn any $n \times n$ partial Latin square into a triangulated tripartite graph G with parts V_1, V_2, V_3 corresponding to the rows, columns and symbols of our Latin square: we do this by associating each filled cell (i, j) containing symbol k to the triangle $\Delta r_i c_j s_k$ in our tripartite graph.



Now, filling in cells in our partial Latin square just corresponds to adding new triangles to our tripartite graph:



In particular, filling in all of the cells of our partial Latin square corresponds to taking all of the edges not yet used in G, and grouping them into triangles:

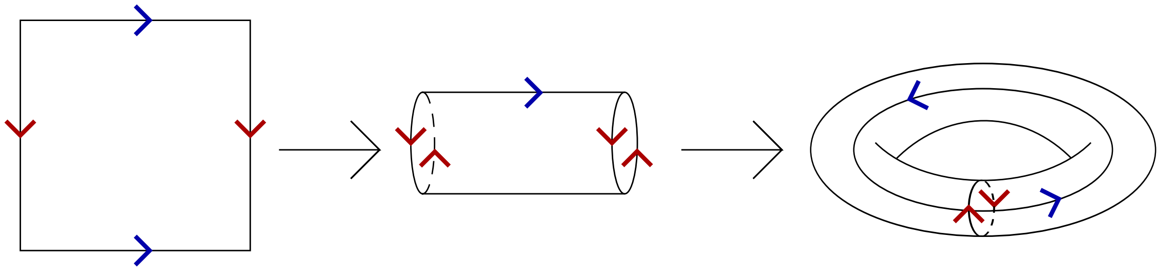


Therefore, these problems are equivalent! We have created a way to turn instances of one of these tasks into instances of the other task. $\square$

Consequently, if we want to show that 3SAT is reducible to the task of completing an arbitrary partial Latin square, we can just reduce 3SAT to the problem of triangulating these special kinds of graphs! To make our lives easier, we're going to prove something slightly weaker here: we're going to reduce 3SAT to the more general task of triangulating a tripartite graph. In other words, we will show that if we have an algorithm that triangulates tripartite graphs, we can use that algorithm to satisfy boolean formulas. (In fact, the tripartite graphs we work with here **are** these special kinds of graphs, but it's tedious to show that, so I'm skipping it. Try to prove it if you're interested, though!)

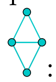
Theorem. 3SAT is reducible to the task of triangulating an arbitrary tripartite graph.

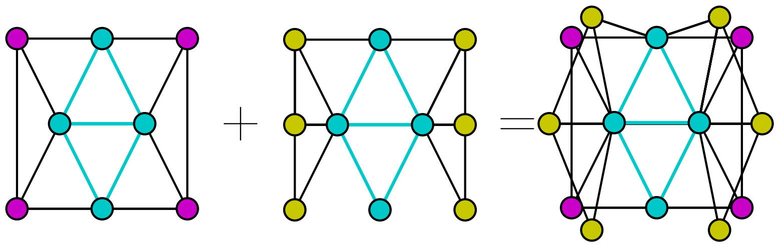
Proof. We first take a quick detour into topology. Specifically, suppose we take a square of some flexible material, like rubber or cloth. Imagine that we take this square, and glue two opposite sides together: this would give us a cylindrical tube. If we were to then glue together the two opposing circles, this would give us a **torus**, or "doughnut," as drawn below:



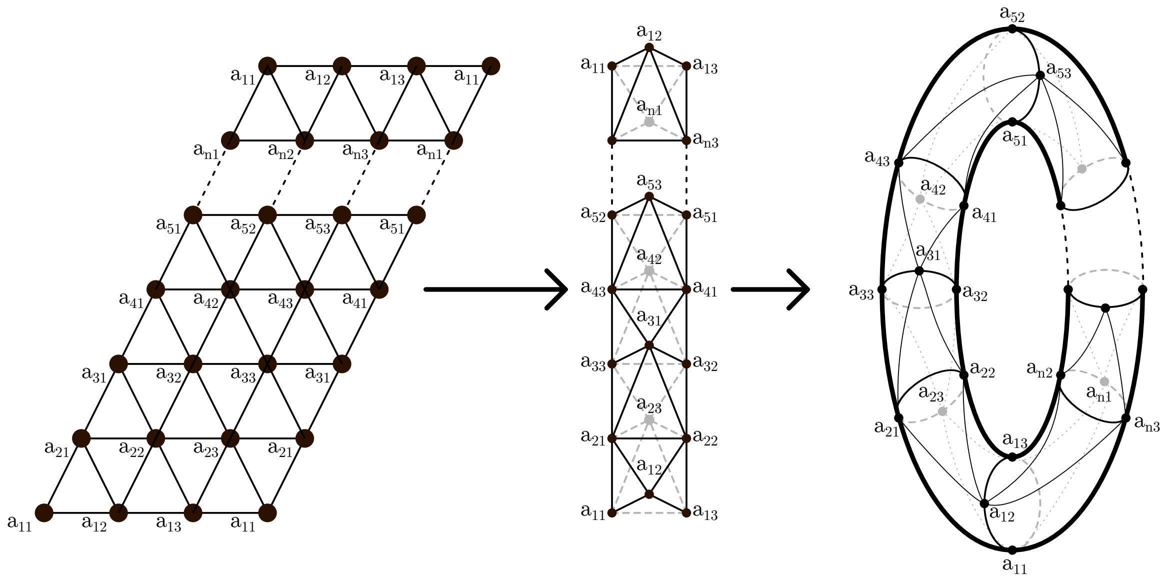
Gluing! We can glue graphs together, as defined here:

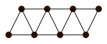
Definition. Take two graphs G_1, G_2. Suppose that they both contain a copy of the same subgraph H. We can **glue** G_1 to G_2 along H by taking our two graphs, and declaring that the copy of H that they both contain is the "same:" i.e. we identify the copy of H in the first graph with the copy of H in the second graph, so that these two graphs are now joined along H!

We illustrate this with an example below, where we glue two different graphs G_1, G_2 along their common subgraph $H =$ :



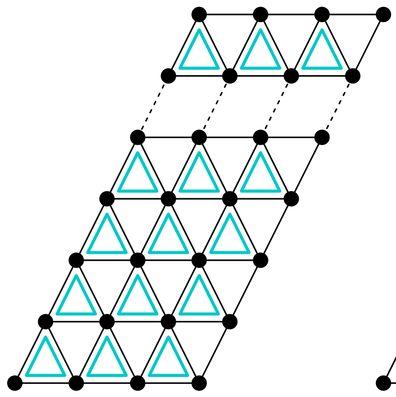
We combine these two ideas together with the following graph, which we call $H_{3,n}$:

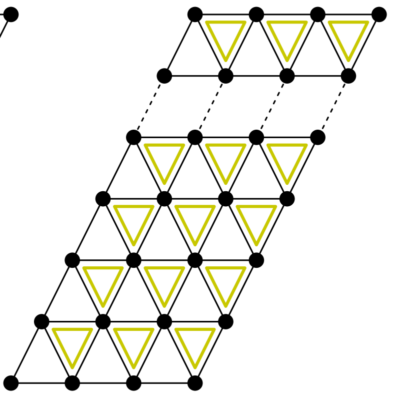


To form $H_{3,n}$, first create the lattice of triangles at the left, which we get by gluing n copies of the graph  to each other. Now, take this lattice and perform the same gluing operation we did to create a torus: i.e. glue the top edge to the bottom edge, and the left edge to the right edge. In the picture above, we indicated how this is done with our vertex labelings: i.e. when we have two different vertices labeled a_{12}, that means that we are considering those vertices to be the **same**! If we do this, we get the "triangulated torus" at the right by actually sticking this left edge to the right edge, and the top edge to the bottom edge.

In practice, we will usually draw $H_{3,n}$ by using the picture at the left, because it's easier to work with the "unrolled" torus and keep track of the fact that we're gluing two edges together, than to have to actually work on a torus all of the time.

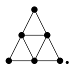
The first reason we care about these graphs is the following property: for any $H_{3,n}$ with $n > 3$, there are only two triangulations on it! We illustrate this in the following picture:


(a true triangulation)

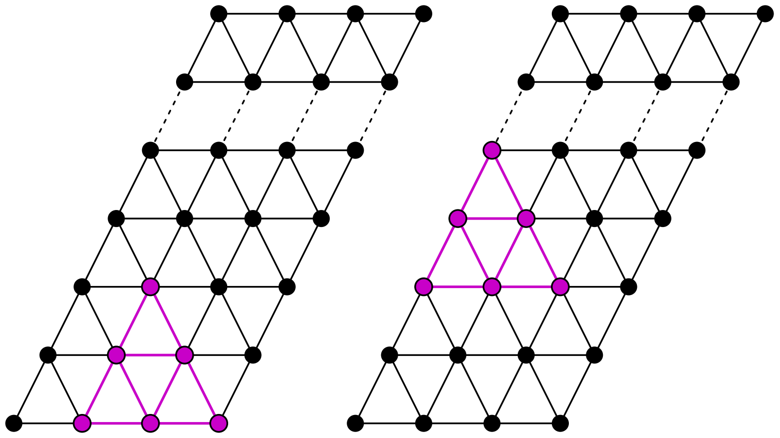

(a false triangulation)

Call a triangulation of the first kind, using Δ triangles, a "true" triangulation, and a triangulation of the second kind, using ∇ triangles, a "false" triangulation.

The second reason we care about these graphs is because they do "interesting" things when glued together in certain ways. Specifically, suppose that we take two $H_{3,n}$ graphs and perform the following "gluing" operation:

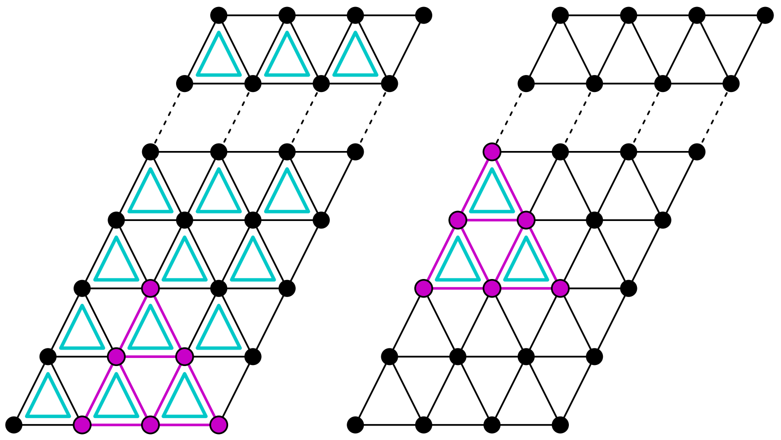
- In each of our two $H_{3,n}$ graphs, find a subgraph in the form the configuration . In particular, make sure that this subgraph has the Δ orientation, with its base at the bottom, in both graphs.
- Glue these two $H_{3,n}$'s together along these six vertices.

What is now true about triangulations of this resulting graph? Well: suppose we have two such graphs glued together.

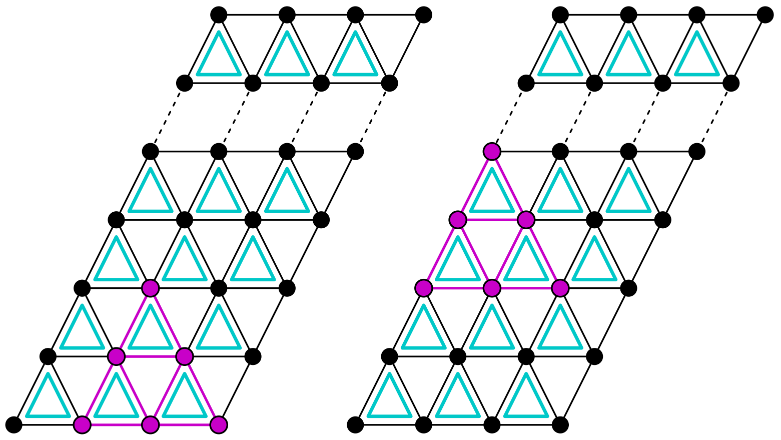


Let's try to start triangulating this graph, starting from the left. As before, the leftmost part has only two triangulations: if we place one Δ triangle, we are forced to only use Δ triangles throughout this entire graph, and similarly for using ∇ triangles. But what does this mean about our second graph?

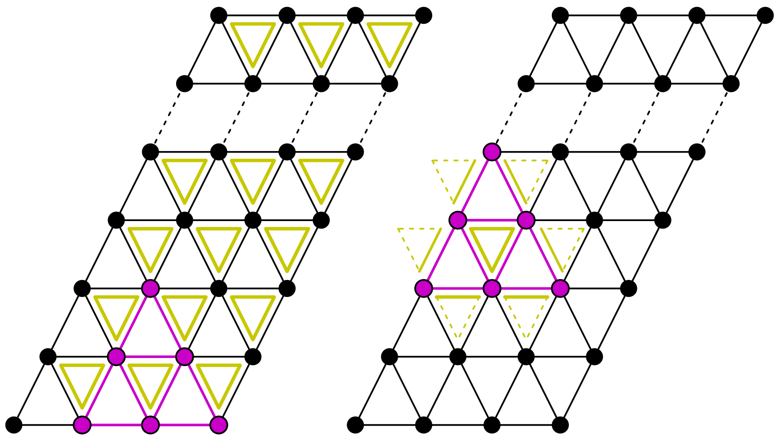
Well: if we used only Δ triangles, we are in the situation



In other words, we've already started using Δ triangles on the right! This forces us to continue using these Δ triangles through the entire right graph:

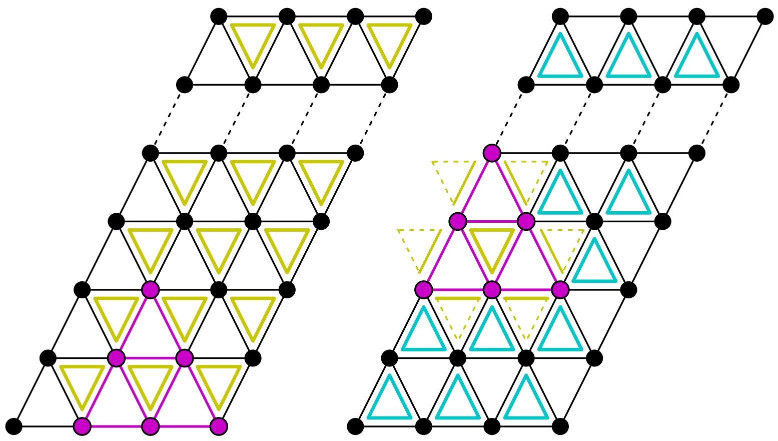


Therefore, we have shown that if we ever start using a Δ triangulation on one of our graphs, we are forced to continue this triangulation through to the other graph. So: what happens if we start by using ∇ triangles? Well: initially, we get

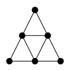
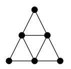


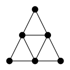
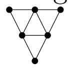
Now: notice that in the right graph, we've started by using all of the purple edges and no other edges. In other words, on the right, it is as if we have already placed three Δ

triangles to use up all of the purple edges! Essentially, this means we've already started to use Δ triangles on the right, and have no choice but to continue:



So: if we start with ∇ triangles on one side, we must use Δ triangles on the other side! Essentially, we have proven the following result:

Lemma. Suppose that we have two $H_{3,n}$'s connected by gluing a  in one graph to a  in the other, so that both patches have their base at the bottom. Then either both of these graphs have true triangulations, or exactly one has a true triangulation and the other has a false triangulation.

For shorthand, we call any subgraph  where the base is at the bottom an A-patch, and any subgraph of the form  where the base is on the top a B-patch.

By using the same logic, it is not hard to get the following corollary to our result:

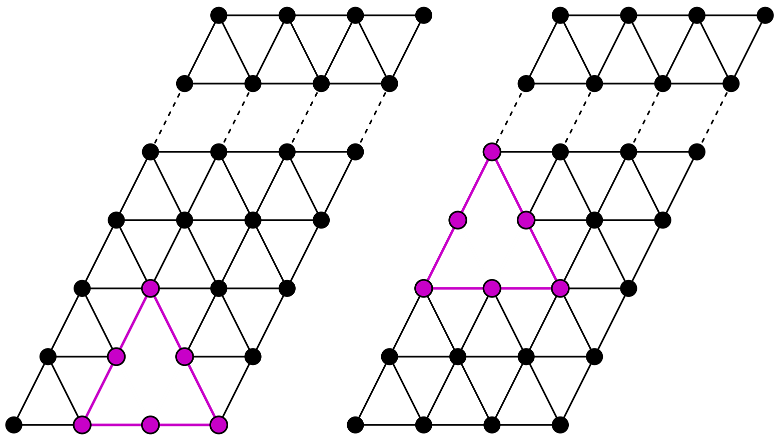
Lemma. Suppose that we have two $H_{3,n}$'s connected by gluing a A-patch in one graph to a B-patch in the other graph.

Then either both of these graphs have false triangulations, or exactly one has a true triangulation and the other has a false triangulation.

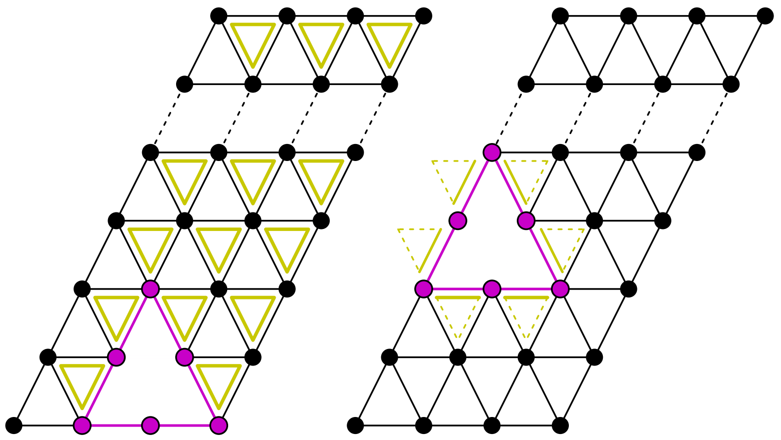
These graphs do interesting things when glued together in other ways:

Lemma. Suppose that we have two $H_{3,n}$'s connected by gluing an A-patch in one to an A-patch in the other. Suppose that after we glue these two graphs together we remove the middle triangle from our A-patch from both graphs. Then exactly one of our graphs has a false triangulation and the other has a true triangulation.

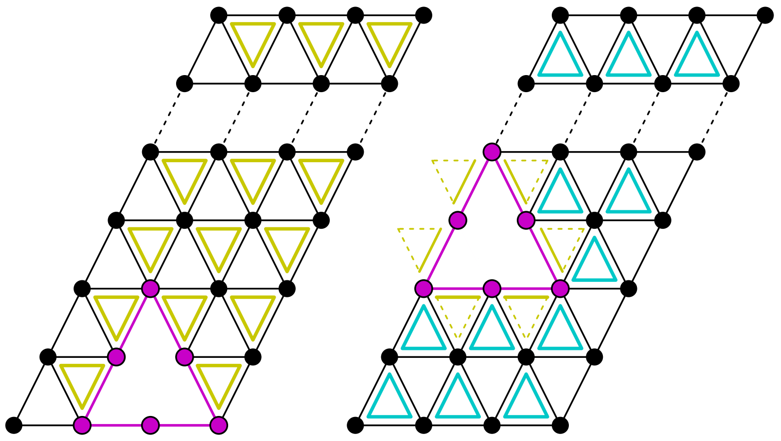
Proof. This is not too hard to see. Again, take any such pair of graphs, and glue them together as directed:



Now, as before, try to triangulate the left-hand-side. Because the middle of our purple triangle is missing, we have in some sense "already picked" the ∇ triangulation, and are forced to use it throughout the entire left-hand side:



As before, this forces us to use the Δ triangles on the right-hand-side.



In other words, exactly one of these graphs has a false triangulation, and the other has a true triangulation, as claimed! $\square$

Again, by using the same logic, it is not hard to get the following result:

Lemma. Suppose that we have several $H_{3,n}$'s connected by finding an A-patch from each graph and gluing them all together along that A-patch. Suppose further that after we glue these two graphs together, we remove the middle triangle from our A-patch from all of these graphs. Then exactly one of our graphs has a false triangulation and all of the others have true triangulations.

Ok. So. How does all of this connect to Boolean formulas? Well: suppose we have any of the Boolean formulas you get from 3SAT: i.e. formulas of the form

$$(l_{1,1} \vee l_{1,2} \vee l_{13}) \wedge (l_{2,1} \vee l_{2,2} \vee l_{2,3}) \wedge \ldots \wedge (l_{n,1} \vee l_{n,2} \vee l_{n,3}),$$

where all of the $l_{i,j}$'s are expressions of the form x or $\neg x$, for some Boolean variable x. For example,

$$(x \vee y \vee z) \wedge (\neg x \vee x \vee x) \wedge (a \vee a \vee a)$$

is a formula we could have, with $l_{1,1} = x, l_{1,2} = y, l_{1,3} = z, l_{2,1} = \neg x, l_{2,2} = x, \ldots$ We call these $l_{i,j}$'s **literals**, and their triples $(l_{1,1} \vee l_{1,2} \vee l_{13})$ **clauses**, for shorthand.

Let's turn this formula into a graph! We do this as follows:

1. For each variable x_k that shows up in our formula, create a $H_{3,n}$ and call it C_{x_k}.
2. As well, for each literal $l_{i,j}$ in our formula, create a $H_{3,n}$, and call it $C_{i,j}$.
3. Whenever a literal $l_{i,j}$ consists of a variable x_k, find an A-patch in C_{x_k} and $C_{i,j}$, and glue these two graphs together along this A-patch.
4. Whenever a literal $l_{i,j}$ consists of the negation of a variable $\neg x_k$, find an A-patch in C_{x_k} and a B-patch in $C_{i,j}$, and glue these two graphs together along these patches.
5. Finally, for each clause $(l_{i,1} \vee l_{i,2} \vee l_{i,3})$, find A-patches in all three of the graphs $C_{i,1}, C_{i,2}, C_{i,3}$, glue these graphs together along these A-patches, and delete the central triangle.

(For the above process, pick the n in our $H_{3,n}$'s large enough so that we can make all of these patches not overlap.)

Now: suppose that the above graph has a triangulation! By our lemmas, we know that

1. Each C_{x_k} has either a true or a false triangulation.
2. If a literal $l_{i,j}$ consists of a variable x_k, then the graph $C_{i,j}$ has a true triangulation whenever C_{x_k} has a false triangulation, and can be either true or false whenever C_{x_k} has a true triangulation.
3. If a literal $l_{i,j}$ consists of the negation of a variable $\neg x_k$, then the graph $C_{i,j}$ has a true triangulation whenever C_{x_k} has a true triangulation, and can be either true or false whenever C_{x_k} has a false triangulation.
4. Exactly one of the graphs $C_{i,1}, C_{i,2}, C_{i,3}$ has a false triangulation, and the other two have true triangulations.

So: what does this mean? Well: let's suppose that we're looking at a clause of the form $(l_{1,1} \vee l_{1,2} \vee l_{13}) = (x \vee y \vee z)$, for three variables $x = l_{1,1}, y = l_{1,2}, z = l_{1,3}$.

1. If all of the graphs C_x, C_y, C_z have false triangulations, this forces the three graphs $C_{1,1}, C_{1,2}, C_{1,3}$ to all have true triangulations, which breaks our fourth condition (exactly one graph $C_{i,1}, C_{i,2}, C_{i,3}$ must have a false triangulation.) So this causes a problem with our triangulation.

2. Otherwise, suppose that one of the graphs C_x, C_y, C_z has a true triangulation. Then its corresponding $C_{1,i}$ can have whatever triangulation we want: either a false triangulation or a true triangulation! In particular, this means that if **at least one** of C_x, C_y, C_z have a true triangulation, we can make sure that **exactly one** of the the graphs $C_{i,1}, C_{i,2}, C_{i,3}$ has a false triangulation, which satisfies all of the conditions we wanted above.

3. In other words, these graphs have triangulations if and only if the expression $(x \vee y \vee z)$ evaluates to true!

This generalizes easily to the situation where we have $\neg x$'s in our clauses: our graphs have triangulations if and only if their corresponding clauses can evaluate to true! Therefore, this triangulation can exist only if our boolean formula is satisfiable. Conversely, if our boolean formula is satisfiable, we can use its true/false assignments to pick out triangulations, which will triangulate our whole graph!

This is what we sought to prove in this talk: if we know how to triangulate tripartite graphs, we know how to satisfy boolean functions! in other words, we have reduced 3SAT to triangulating tripartite graphs. $\square$