NP and Latin Squares | Instructor: Padraic Bartlett

Lecture 4: 3SAT and Latin Squares

Week 4 | *Mathcamp 2014*

This talk's focus is on the computational complexity of completing partial Latin squares. Our first goal in this talk is to show that for many special classes of partial Latin squares, we can construct completions for these objects in polynomial time. From here, we will prove the fairly surprising claim that completing an arbitrary partial Latin square is an NP-complete task; so while it is "easy" to complete almost all reasonably-well-behaved families of partial Latin squares, it is remarkably hard to complete any partial Latin square without some constraints on what it can or cannot be.

1 Partial Latin Squares Completable in Polynomial Time

In this part, we give a number of families of partial Latin squares can be completed to proper Latin squares in polynomial time.

To start , let's consider one of the simplest families to complete: Latin rectangles!

Definition. A $k \times n$ **Latin rectangle** is a $n \times n$ partial Latin square, such that every cell in its first k rows is filled, and no cell in its remaining rows is filled.

Theorem. Any Latin rectangle can be completed to a Latin square. Furthermore, this completion can be calculated in polynomial time.

The proof of this result is essentially an application of a famous result of Hall, called **Hall's marriage theorem**:

Theorem. Take any bipartite graph G with bipartition $(V_1, V_2$. Suppose that G has the following property: given any i and set $S \subseteq V_i$, the number of neighbors to elements in S, $N(S)$, is at least as large as the number of elements in S itself. (In symbols: $|N(S)| \geq |S|$. This is called Hall's condition in the literature.)

Then G has a **perfect matching**: i.e. there is some collection M of edges so that every vertex of G shows up in exactly one edge. (In general, a **matching** is any collection of edges so that every vertex shows up in at most edge; matchings are perfect precisely when they contain every vertex.)

We start by proving Hall's marriage theorem:

Proof. We proceed via the following algorithm, that takes in any nonperfect matching M and creates a matching M' with strictly larger size:

1. Because of Hall's condition, the two sides V_1, V_2 must be the same size.
2. Therefore, if M is not perfect, there must be some starting vertex $x_1 \in V_1$ that is not in our matching.

3. By Hall's condition, this vertex must have a neighbor x_2 in V_2. Travel to this vertex along our edge.

4. If the vertex x_2 is not in our matching, then we can increase the size of our matching by simply adding in the edge $\{x_1, x_2\}$ to our matching.

5. Otherwise, it is in our matching, and there is a third vertex x_3 connected to x_2 by a matching edge, that is not x_1. Travel to x_3.

6. Consider the common neighbors of x_1, x_3. There must be some vertex beyond x_2 that we can reach from one of these vertices, by Hall's condition: take this vertex as our new vertex x_4, and travel to this vertex. Note that this edge is not in our matching.

7. If x_4 is not in our matching, then halt. Otherwise it is; use this observation to travel back to some vertex x_5. Again, note that x_5 is not one of our earlier-reached vertices, because M is a matching.

8. Once again, notice that we must be able to get to some new vertex x_6 from x_5, by the same logic as two steps ago! In other words, we can simply loop the logic of steps 6-7; we will never get stuck on step 6 by Hall's condition, and thus we eventually must halt at step 7, as our graph is finite.

What is the output of this algorithm? Well: it is a sequence of vertices that starts from a vertex not in our matching, ends at a vertex not in our matching, and along the way alternates between edges not in our matching and edges in our matching.

So: take this alternating path, and switch which edges are and are not in our matching! This creates a matching with one more edge than we started with, and did so in polynomial time (check this in the HW!) Iterating this process creates a perfect matching, as desired. □

We now use this result to prove our original claim about completing Latin rectangles:

Proof. Take our $k \times n$ Latin rectangle L. Form the following bipartite graph:

- Vertex set 1: the columns of L.
- Vertex set 2: the symbols of L.
- Edges: connect c_j to s_k if and only if symbol k does not show up in column j.

This graph satisfies Hall's condition (why?) Consequently, it has a perfect matching. But what is a perfect matching? Well: it is a way to pair up each column with a different symbol, so that each paired symbol does not occur in that column yet! In other words, this perfect matching corresponds precisely with a potential $k+1$-th row for our Latin rectangle! Add this as a row: we have now grown our Latin rectangle by one row, in polynomial time.

Doing this repeatedly will complete our Latin rectangle to a Latin square, as desired. □

We list a few additional results here, along these lines:

1. (Ryser, 1951.) Suppose that P is a $n \times n$ partial latin square with the following properties:

   - There is a set of r rows named R, and a set of c columns named C, such that a cell (i, j) is not blank iff $i \in R$ and $j \in C$.
   - If $N(k)$ denotes the total number of times the symbol k is used in our entire square, then $N(k) \geq r + c - n$.

   Then P can be completed

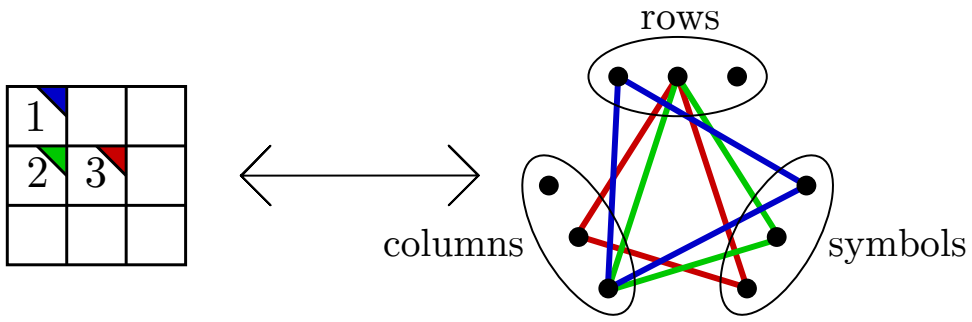
2. (Smetaniuk, 1981.) Suppose P is a partial latin square with $\leq n - 1$ filled cells. Then P can be completed.

3. (Buchanan, 2007.) Suppose P is a $n \times n$ partial Latin square consisting of 2 filled rows and columns of P are . Then P can be completed whenever $n \geq 6$.

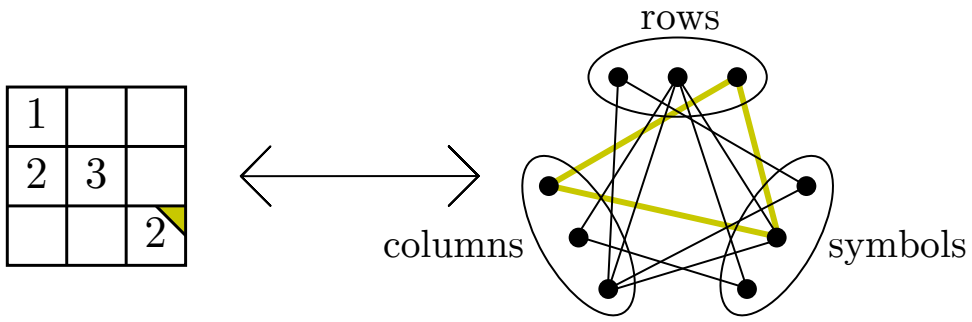
All of the results above consist of algorithms that complete these squares; all of these algorithms run in polynomial time (with the caveat that I haven't read carefully through Buchanan's 140+ page thesis yet, but it certainly looks like it runs in polynomial time.)

2 Completing Arbitrary Partial Latin Squares

Given the above results, it may be surprising that completing an arbitrary partial Latin square is a NP-complete task; after all, every time we consider a specific family of Latin squares, they are all individually completable! However, given yesterday's lecture, this doesn't seem too implausible. Consider our correspondence between triangulated tripartite graphs and Latin squares:



Under this correspondence, adding cells to our grid corresponds to adding triangles to our graph! Therefore, completing our partial Latin square is equivalent to triangulating the complement of some triangulated tripartite graph:



This correspondence gives us the following reduction for free:

Corollary. The problem of completing a partial Latin square reduces to the task of triangulating a graph.

However, this is in a sense the "wrong direction." We want to show that completing partial Latin squares is **hard**: in other words, we want to reduce some NP-complete problem, like triangulating graphs, to completing partial Latin squares!

To do this, then, we don't want a way to embed Latin squares in graphs; we want a way to embed graphs into Latin squares! If done directly, this is overly difficult; arbitrary graphs are very complicated things. However, if we look at our proof from yesterday, we can notice a few particularly nice properties about the graphs $H_{m,n}$ we dealt with:

1. If we picked n, m to both be multiples of 3, this graph is tripartite!
2. Moreover, if n, m are chosen so that our graph is tripartite, the size of each part is the same.
3. This graph is **uniform** — in other words, given any vertex v in a part V_i of these tripartite graphs, the number of edges from this vertex to the $i+1$-th part of our graph is the same as the number of edges from this vertex to the $i-1$-th part of our graph. (I.e. $\deg_{i+1}(v) = \deg_{i-1}(v)$).
4. These properties are preserved under the gluing operations we defined.

(Proofs of these claims are on the homework!)

In particular, this means that we can slightly modify our claim from yesterday

Theorem. 3SAT reduces to the task of triangulating a uniform tripartite graph on (n, n, n) vertices.

This is stronger: i.e. we no longer need an algorithm that triangulates arbitrary graphs to solve 3SAT problems, we just need one that can triangulate uniform tripartite graphs! In turn, this makes our work easier for dealing with Latin squares: we just need a way to turn any uniform tripartite graph into a partial Latin square, such that completions of the corresponding partial Latin square turn into triangulations of the tripartite graph!

This is possible, as Colbourn showed in 1984:

Theorem. (Colbourn.) Triangulating a uniform tripartite graph is a task that reduces to completing partial Latin squares.

Colbourn's proof is a relatively straightforward one, that mostly hinges around combining Hall's marriage theorem with the following concept:

Definition. Take any uniform tripartite graph G with parts (R, C, S), where $R = \{r_i\}_{i=1}^n, C = \{c_i\}_{i=1}^n, S = \{s_i\}_{i=1}^n$. A **Latin framework** $LF(G; x, y, z)$ is an $x \times y$ array with the following properties:

1. Each cell of our array is either blank, or contains a symbol from $\{1, \dots z\}$.
2. There are no repeated symbols in any row, column, or symbol.
3. If the edge (r_i, c_j) exists in the graph G, then the cell (i, j) in our array is blank; otherwise, it is filled.
4. If the edge (r_i, s_k) exists in the graph G, then there is no symbol k in the i-th row of our array.
5. If the edge (c_j, s_k) exists in the graph G, then there is no symbol k in the j-th column of our array.

Notice that when $x = y = z$, our framework is a partial Latin square, as it is an $x \times x$ array filled with blanks and the symbols $\{1, \dots x\}$, with no repeats. Furthermore, this partial Latin square corresponds precisely to the tripartite complement of the graph G: when translated to a graph, its edges are precisely those edges that G does not have. Consequently, any completion of this partial Latin square would correspond to a triangulation of G itself!

With this stated, our proof has a fairly natural structure. Take any uniform tripartite graph G. Then:

1. Form a corresponding Latin framework $LF(G, x, y, z)$ in polynomial time.
2. Extend this Latin framework to a larger Latin framework $LF(G, w, w, w)$, for some value w, again in polynomial time.
3. By the above comments, this Latin framework corresponds to a partial Latin square; completing it triangulates our graph G. So, if we had any algorithm for completing a partial Latin square, we could use it here to complete our framework, and thus triangulate G itself.

The first step of this is relatively simple. Given a uniform tripartite graph G on (n, n, n) vertices, consider the following $n \times n$ array:

- If (r_i, c_j) is an edge in G, leave the cell (i, j) blank.
- Otherwise, fill it with the symbol $(i + j \mod n) + 1 + n$.

By construction, this satisfies all of the properties of a $LF(G, n, n, 2n)$, mostly because we added in n additional symbols and are using them as placeholders through our entire array! But hey, it works.

So, all we need to do is describe how to go from a $LF(G, n, n, 2n)$ to a $LF(G, 2n, 2n, 2n)$. We do this via the following lemma:

Lemma 1. *Take any Latin framework $L = LF(G; x, y, z)$ for a uniform tripartite graph G. For any symbol k, let $R(k)$ denote the number of times that k occurs in L, plus the total number of edges in the graph G from s_k to the row-set R; this number, in a sense, corresponds to the number of rows that k "cannot be used in" throughout our Latin framework.*

Suppose that for every symbol k, we have

$$R(k) \geq x + y - z$$

Then we can expand our Latin framework by one column, in a way that preserves the above inequality. In other words, we can extend our Latin framework L to some $L' = LF(G; x, y + 1, z)$, such that L and L' agree on all of L.

Proof. For each row i, define the set S_i as the collection

$$S_i = \{k \mid k \notin \text{row } i, \text{ and } (r_i, s_k) \notin E(G)\}.$$

In other words, S_i corresponds to the collection of all symbols k that are possible candidates for the i-th entry in our new $y + 1$-th column.

Notice that each set S_i contains precisely $z-y$ elements. This is because every cell (r_i, c_i) in some row i is either filled (in which case that symbol corresponds to exactly one element that cannot be in our row) or is blank (in which case (r_i, c_i) is an edge in G, and therefore because of uniformity there is a corresponding (r_i, s_k) that is blocked from occurring in this row as well.) Consequently, the number of possible symbols is exactly the total number of possible symbols, minus the number of cells in our row: i.e. $z - y$.

As well, define the set

$$M = \{k \mid R(k) = x + y - z\}.$$

This set corresponds to the symbols in our Latin framework that are "close" to being used too rarely for our lemma to work.

Given these sets, if we want to construct a new column $y + 1$ for our Latin framework, we need to simply do the following:

1. Pick one element from each S_i, so that no element is repeated across all of our choices. (This constructs our new column.)

2. While making these choices, we should insure that we pick all of the elements of M in constructing our new column; this will "use" each of these symbols in M at least once, and thus preserve the "$R(k) \geq x + y - z$" property we want our graphs to have.

Both of these results are consequences of the following slight modification of Hall's marriage theorem:

Theorem. (Hall.) Take any bipartite graph $G = (V_1, V_2)$ that satisfies Hall's condition on subsets of V_1: that is, given any subset $S \subset V_1$, we have $|S| \leq |N(S)|$. Then there is a matching in G that uses all of the vertices in V_1.

The proof of the above is identical to the proof we presented for Hall's theorem earlier in these notes!

Gven this, consider the following bipartite graph:

- Vertex set 1: the sets $\{S_i\}_{i=1}^{x}$.
- Vertex set 2: the symbols $\{k\}_{k=1}^{z}$.
- Edges: connect S_i to k if and only if $k \in S_i$.

A matching of the sets $\{S_i\}_{i=1}^{x}$ to the symbols then corresponds to a new potential column, which is exactly what we're looking for! Therefore, it suffices to construct such an object, which we can do via Hall's marriage theorem if and only if our graph satisfies Hall's criterion.

We check if our construction satisfies Hall's criterion. Take any subset $A \subseteq \{S_i\}_{i=1}^{x}$. On one hand, by our discussion above, the degree of each set-vertex S_i is $z - y$, because each set contains $z - y$ elements.

On the other hand, consider any symbol k in our Latin framework; by assumption, it is "blocked" from occurring in at least $R(k) \geq x + y - z$ of the rows of our square. Consequently, there are at most $x - (x + y - z) = z - y$ rows in which k can occur, for any k! In other words, the degree of each symbol-vertex k is at most $z - y$.

Therefore, if we have $|A|$ of the S_i's, we have $|A| \cdot (z - y)$ edges leaving our set A; if the degree of every symbol-vertex is at most $z - y$, we need at least $|A|$ symbol-vertices to absorb the $|A| \cdot (z - y)$ incoming edges! In other words, our graph satisfies Hall's condition, and thus has a matching using all of the S_i's, as desired.

This gives us a new column; however, we also need to insure that this new column contains all of the elements of M! We do this through another similar matching exercise. Consider the following graph:

- Vertex set 1: the set M of symbols.
- Vertex set 2: the sets $\{S_i\}_{i=1}^{x}$.
- Edges: connect k to S_i if and only if $k \in S_i$.

We now want a matching of M to the S_i's, so that every element of M gets matched to some S_i. We do this by again checking Hall's condition. On one hand, we know that by the definition of M, each symbol $k \in M$ is blocked from occurring in $R(k) = x + y - z$ symbols, and thus is in **exactly** $x - (x + y - z) = z - y$ sets S_i. In other words, each symbol k in our graph has degree $z - y$. As discussed before, each set S_i in our set contains $z - y$ elements, and thus has degree $\leq z - y$ in the graph we have constructed here. Therefore, by the exact same logic as before, any subset A of M has $|A|(z - y)$ edges leaving it, and therefore must have at least $|A|$ neighbors across from it in $\{S_i\}_{i=1}^{x}$. In other words, Hall's condition holds, and we have a matching!

We're now almost done. We have, at this point, the following:

- A matching from $\{S_i\}_{i=1}^{x}$ to $\{1, \ldots z\}$ that uses all of the S_i's.
- A matching from M to $\{S_i\}_{i=1}^{x}$ that uses all of the M's.

What we want is, in a sense, "both" of these matchings: we want a matching from $\{S_i\}_{i=1}^{x}$ to $\{1, \ldots z\}$ that uses all of the S_i's, but that **also** hits all of the M's! Conveniently, this is possible, as the following theorem shows:

Theorem. Take any bipartite graph $G = (V_1, V_2)$ with matchings M_1, M_2. There is a matching $M \subseteq M_1 \cup M_2$ such that the following holds:

- Every vertex in $V_1 \cap M_1$ is contained in M, and also
- Every vertex in $V_2 \cap M_2$ is contained in M.

We prove this theorem here:

Proof. Take any such graph G and matchings M_1, M_2. Look at the subgraph of G given by taking the edges/vertices given by $M_1 \cup M_2$: this is a graph in which every vertex has degree at most 2, and therefore (because G is bipartite) decomposes into a collection of paths and even cycles.

Take any of the connected components H_i of $M_1 \cup M_2$, and do the following:

- If H_i is a cycle, then take every other edge from it and put those in M. Notice that this choice covers every vertex that H_i hit, so we are still covering the same sets of vertices that $M_1 \cup M_2$ covered.
- Suppose that H_i is a path, and moreover contains a vertex in V_1 that is in M_1 and not in M_2. Then our path must start at this vertex! Take all of the M_1-edges in H_1. This trivially covers all of the vertices in V_1 hit by M_1; to see that it also covers all of the V_2 vertices hit by M_2, simply observe that because our path starts in V_1 and travels to V_2 along a M_1-edge, it must always go from V_2 to V_1 along M_2-edges, and thus in particular can't end on a V_2-vertex reached by a M_2-edge.
- Identical reasoning to the above tells us that if H_i is a path that contains a vertex in V_2 that is in M_2 and not in M_1, taking the M_2-edges will always work out for us.

Because any H_i falls into one of the three cases above, we have described how to create a matching with the desired properties. This proves our lemma, as desired. □

Applying this theorem to the matching M_1 from $\{S_i\}_{i=1}^x$ to $\{1, \dots z\}$ and the matching M_2 from M to $\{S_i\}_{i=1}^x$ proves our lemma! So we are done.

Furthermore, notice that this lemma can be applied in polynomial time; its steps consist of using Hall to make matchings (doable in polynomial time) and then of combining these matchings (again doable in polynomial time.) □

From here, our theorem — that triangulation reduces to completing a partial Latin square — is relatively straightforward. Simply take our tripartite uniform graph G, and turn it into a $LF(G; n, n, 2n)$ as discussed earlier. Apply the lemma above repeatedly to turn it into a $LF(G; n, 2n, 2n)$; then take the transpose of this array, and repeatedly apply the lemma again until we get a $LF(G; 2n, 2n, 2n)$. As discussed earlier, this is precisely a partial Latin square whose complement (interpreted as a tripartite graph) is G. Therefore, if we have any algorithm that can complet partial Latin squares, we can use the above polynomial-time process to use it to triangulate uniform tripartite graphs!

This reduces triangulation to completing a partial Latin square, as desired. In particular, notice that this result in conjunction with Wednesday's proof that triangulation is a NP-complete problem proves that completing an arbitrary Latin square is NP-complete!