6.890: Algorithmic Lower Bounds: Fun With Hardness Proofs	Fall 2014
Lecture 9 — October 2, 2014	
Prof. Erik Demaine	*Scribe: Nathan Pinsker, Patrick Yang*

1 Overview

In this lecture, we will look at a variety of graph-related problems and other miscellany which may be helpful but have not fit into any topic from the past lectures. In particular, we will look at problems involving *vertex covers*, *coloring*, and *ordering.* We also look at problems involving *orientations* of graphs. We conclude by looking at graph *crossing number* and *Rubik's cubes.*

2 Vertex Cover

Recall that a *vertex cover* is a set of k vertices such that each edge in a graph is adjacent to at least one vertex in the set. Lichtenstein showed in [10] that planar vertex cover is NP-hard by reducing from planar 3SAT. This result holds even for graphs with maximum degree 3.

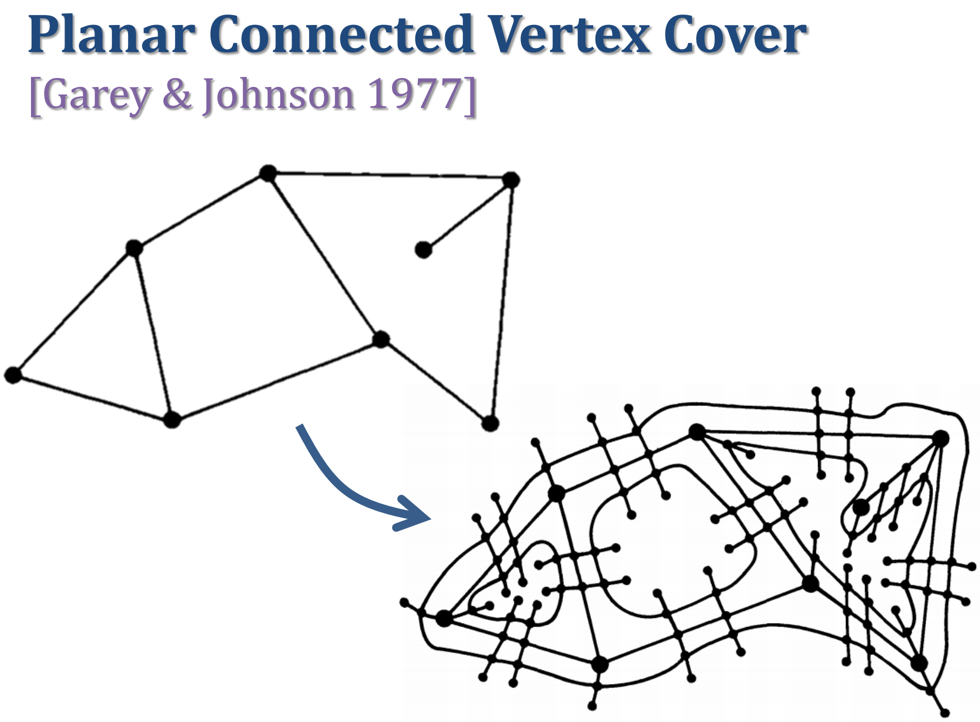
Note that vertex cover can be interpreted as a 2SAT problem on a graph, where we must choose exactly k vertices (in other words, there will be exactly k true variables).

We start by briefly mention two related problems, both of which are solvable in **polynomial time**:

- **Exact vertex cover**, where each edge must be incident to exactly one vertex.
- **Edge cover**, where we choose k edges to cover all vertices in a graph.

2.1 Planar connected vertex cover

In this variation of a vertex cover, we require the chosen vertices to induce a connected subgraph. This problem is NP-hard, and we prove this by reducing from planar vertex cover. Our reduction will hold for planar subgraphs with maximum degree 4.



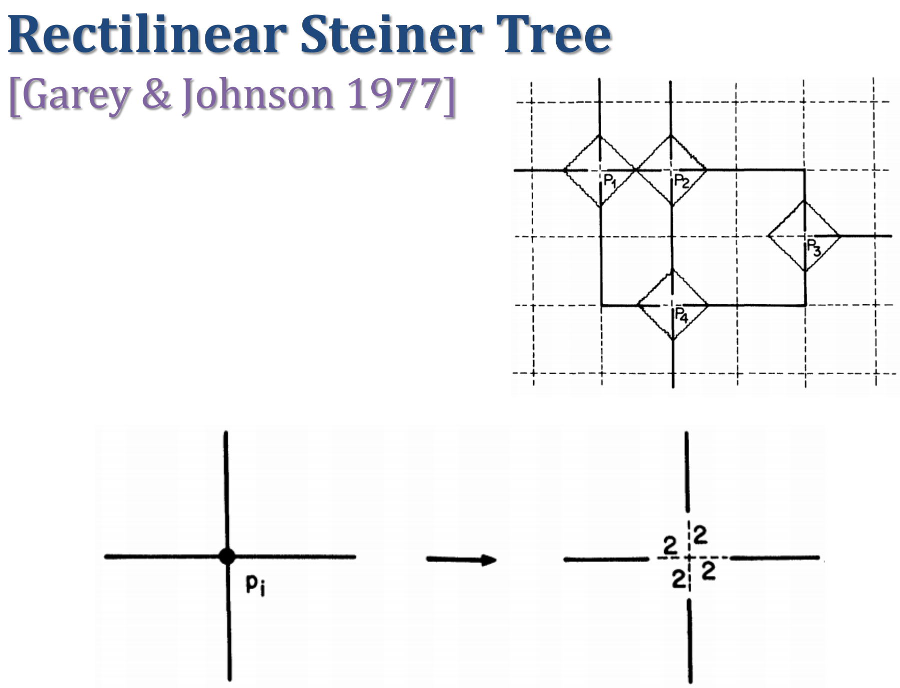
In the diagram above, we transform our given planar graph G by adding a closed loop for each face in the graph. For each edge in G that separates two loops l_1 and l_2, we add several vertices that "connect" the two closed loops together. The construction adds exactly $5 \cdot |E|$ edges to the graph, and increases each of the original vertex's degrees up to 4.

Note that there always exists an optimal vertex cover where we never choose any leaves in the graph. This is because choosing the node adjacent to a leaf in our cover is always at least as good as choosing the leaf itself. Thus, to obtain a connected vertex cover, we must choose exactly one of the two nodes in each subdivided edge to connect each of the closed loops together. (It is never more useful to choose both, since we could simply choose a vertex of our original graph G and be guaranteed to cover at least as many nodes.) After we have done so, these additions induce a connected graph.

3 Rectilinear Steiner Tree

Given n points in the plane, the *Steiner tree problem* asks for the minimum length of "road" needed to connect all the points together. Note that we are allowed to add arbitrary vertices to shorten the length of road. For example, if we are given 4 points at $(-1,-1), (-1,1), (1,-1), (1,1)$, then we can add a vertex at $(0,0)$ and connect each of our four given vertices to the new vertex to form the minimum Steiner tree.

In the rectilinear Steiner tree problem, all roads must be parallel to one of the coordinate axes. We can prove this problem is NP-hard by reducing from the connected vertex cover problem, as in [7].

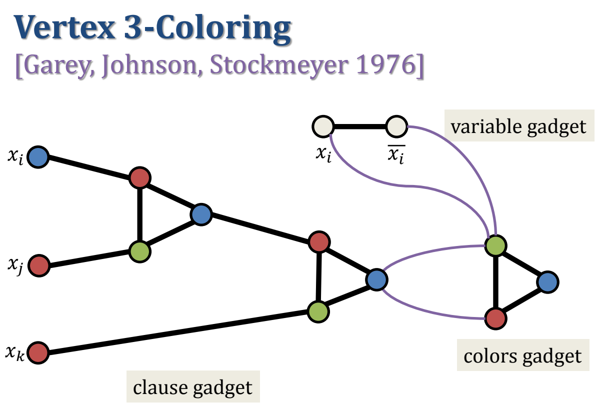


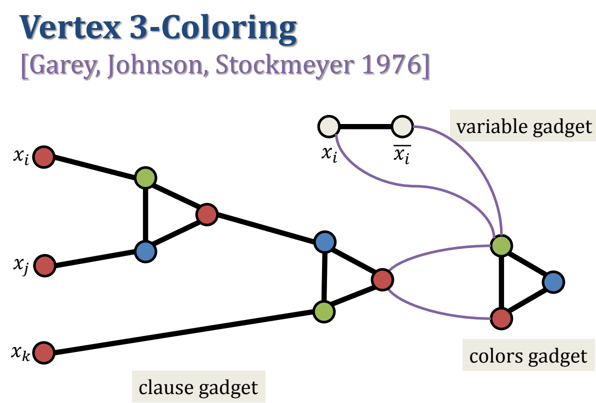
We draw our given graph rectilinearly on the grid, and scale it by $4n^2$. We add auxiliary points at all integer points along the edges, except within radius 1 of the vertices. (The vertex transformation is shown in the diagram above.) It is fairly easy to see that each of the points comprising the "edges" of the graph must be connected to its adjacent points. Also, every edge must connect to a vertex, so we must add at least $2|E|$ edges. We must also connect the other end of the edges in a spanning tree of G, which means we must add at least $2(|V|-1)$ edges.

4 Vertex k-coloring

Given a graph and a positive integer k, the vertex k-coloring problem is to find whether a color assignment $c : V \to \{1, 2, \ldots, k\}$ exists, such that no edge (v, w) has $c(v) = c(w)$.

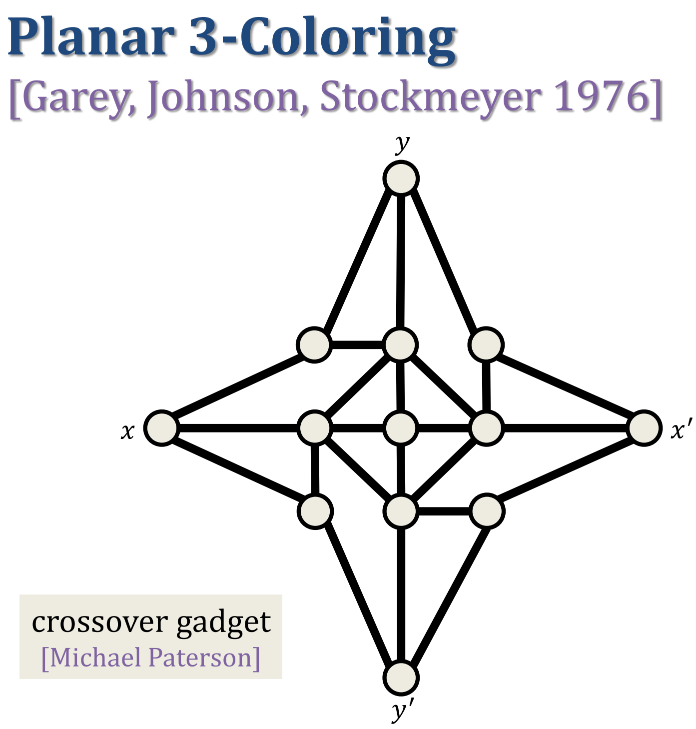
Checking whether a 2-coloring exists (bipartiteness) is doable in polynomial time. However, [8] shows that checking whether a 3-coloring exists is NP-hard by reducing from 3SAT. We do so by constructing variable, clause, and "colors" gadgets. In the diagrams below, the blue color represents true, and the red color represents false. The only bad case is where the rightmost node of the clause gadget must be red, meaning the graph cannot be colored. This occurs only when all x_i are red.



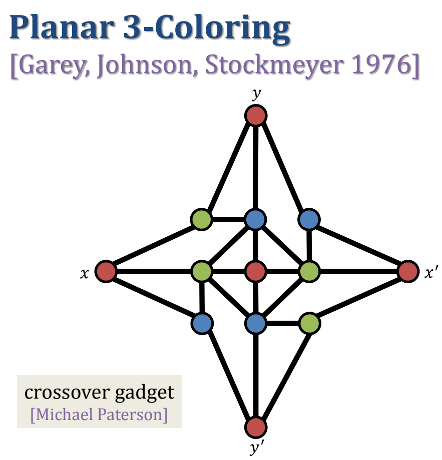


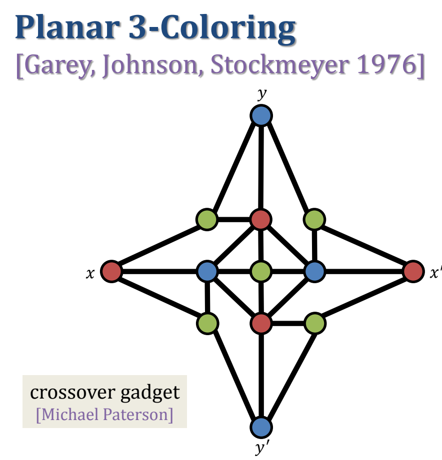
4.1 Planar 3-coloring

We can show that planar 3-coloring is NP-hard by reducing to vertex 3-coloring. We only need present a "crossover" gadget, which we do below.



Below are the two distinct ways (up to permutation) in which the crossover gadget can be colored. Both cases result in $x = x'$ and $y = y'$.

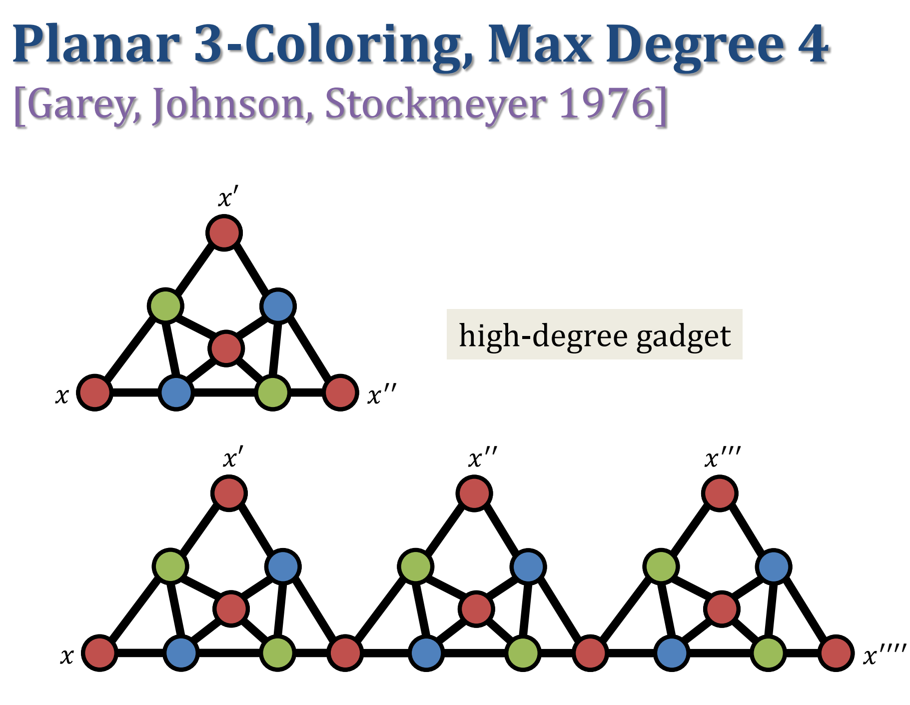




There is one subtlety in the use of the crossover gadget. Say the edge from x to z included a crossing, and we wished to use the above crossover gadget. We would **NOT** identify x' with z, as we wish those vertices to be different colors. Instead, draw a segment connecting x' to z.

4.2 Planar 3-coloring, maximum degree 4

We show that this is NP-complete by reducing to the previous problem of planar 3-coloring. We construct a "high degree" gadget, which we use to emulate vertices with degree larger than 4.



Note that 3-coloring with maximum degree 3 can be solved in polynomial time, even when the graph is not planar. This is because coloring the graph is always possible unless the graph is K_4 (or, if the graph is not connected, contains K_4 as a component). This result is known as Brooks'

Theorem [1].

5 Pushing 1×1 blocks

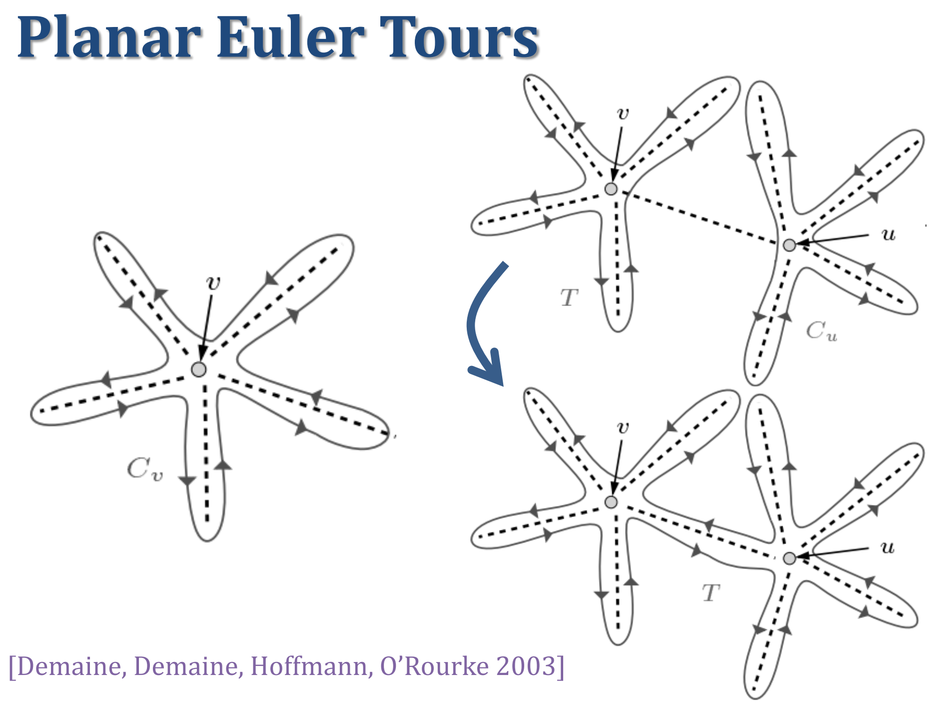
We start with an overview of the known complexities of some variations of the Push-k puzzle, and then proceed to show the NP-hardness of some of these varities.

Pushing 1×1 Blocks Complexity

Name	Push	Fixed	Slide	Goal	Complexity	Reference
Push-k	$k \geq 1$	no	min	path	NP-hard	D, D, O'Rourke 2000
Push-$*$	∞	no	min	path	NP-hard	Hoffmann 2000
PushPush-k	$k \geq 1$	no	max	path	PSPACE-complete	D, Hoffmann, Holzer 2004
PushPush-$*$	∞	no	max	path	NP-hard	Hoffmann 2000
Push-1F	1	yes	min	path	NP-hard	DDO 2000
Push-kF	$k \geq 2$	yes	min	path	PSPACE-complete	D, Hearn, Hoffmann 2002
Push-$*$F	∞	yes	min	path	PSPACE-complete	Bremner, O'Rourke, Shermer 1994
Push-kX	$k \geq 1$	no	min	simple path	NP-complete	D, Hoffmann 2001
Push-$*$X	∞	no	min	simple path	NP-complete	Hoffmann 2000
Sokoban	1	yes	min	storage	PSPACE-complete	Culberson 1998

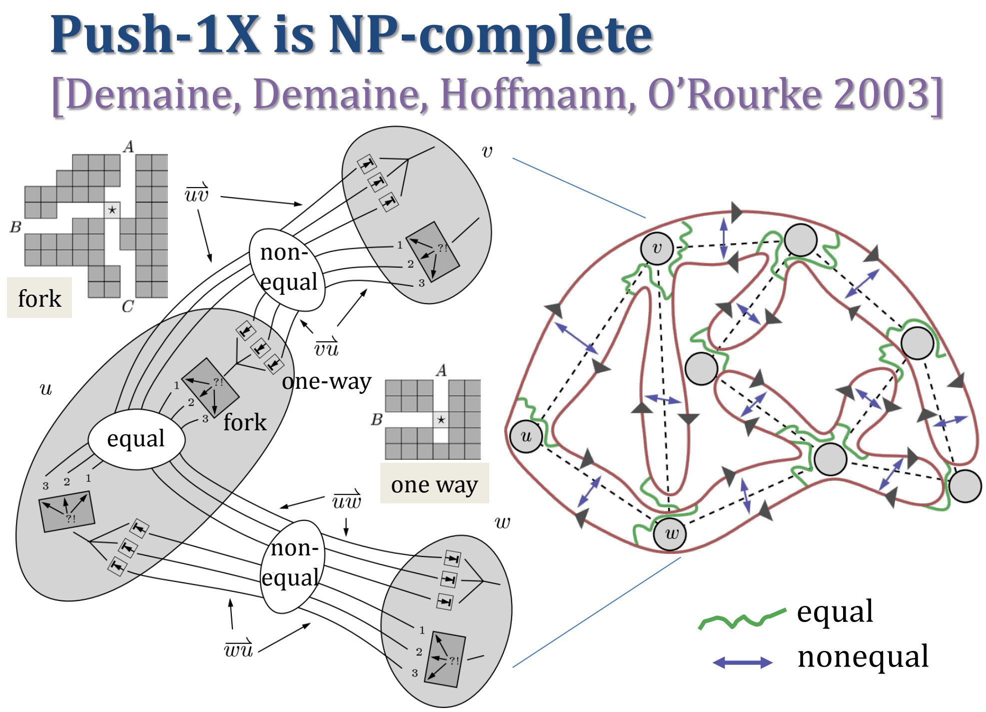
5.1 Planar Euler tour

We use the existence of a *planar Euler tour* in the upcoming proof. Such a tour visits all the edges of a planar graph in a "planar" way: in other words, it visits all the edges of each vertex in a clockwise order, and does not cross its own path. Such a tour's existence can easily be proven and constructed inductively as in the following diagram. On the right half of this diagram, we see a graph such that two disjoint subgraphs have planar Euler tours, and the two subgraphs are connected by an edge. We run the planar Euler tours 'around' this edge, thus generating a planar Euler tour for the entire graph.

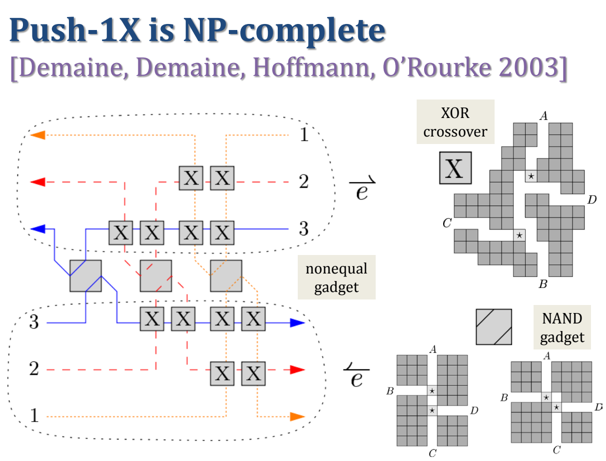


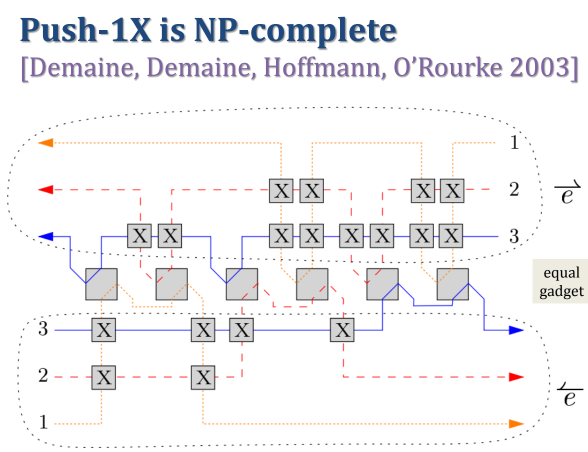
5.2 NP-completeness of Push-1X

The Push-1X problem is the problem of whether we can get to a specified location by pushing blocks in a grid, subject to the stipulation that we never visit the same location twice. We show this is NP-complete by reducing from planar 3-coloring with maximum degree 4; the reduction is due to Demaine et al. [3] We start by constructing a planar Euler tour of the graph, and choose the color of each vertex whenever we visit it. We choose the color along our tour by using a "triple" planar Euler tour, in which we split the path into three paths, each one corresponding to a chosen color. We also construct "one-way" gadgets, so that we can forget the color we have chosen for the next vertex we visit.



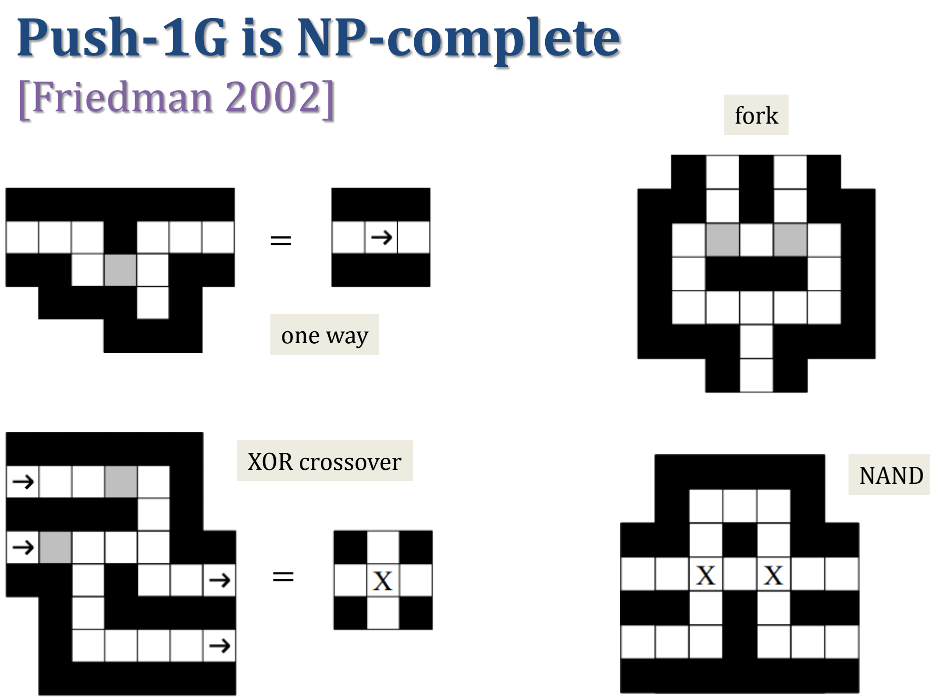
We also construct gadgets that force equality and non-equality at two vertices. These gadgets can be reduced to XOR crossovers and NAND gates (two adjacent paths, only one of which can be used). Whenever we enter the area around a vertex, we also enter an "equal" gadget to ensure each vertex only has one selected color. However, on the two paths going in both directions on each edge, we bind them together with a "nonequal" gadget. The two paths in each direction will correspond to the two colors chosen at the endpoints; thus, a "nonequal" gadget will ensure the 3-coloring condition holds.





5.3 NP-completeness of Push-1G

The Push-1G problem is We again reduce to this problem from planar 3-coloring with maximum degree 4. The reduction is due to Friedman [5]. As before, we construct a one-way gadget, a fork gadget, and an XOR crossover gadget. The construction of these gadgets is simpler (as we are allowed to revisit squares), and is shown below.



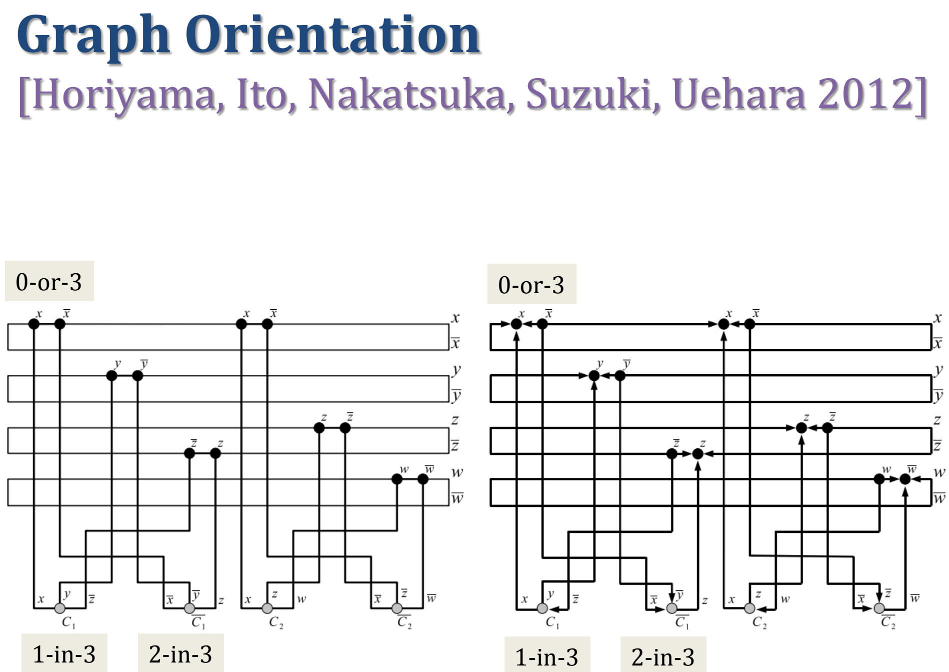
In the one way gadget above, the gray block falls down when approaching from the left, but will not be passable when approaching from the right. In the XOR crossover, traversing either path will also disable the other path due to a falling block. The NAND gadget is constructed out of two XOR crossovers (in fact, this could have been done for the Push-1X proof above as well, but that construction is presented as it was performed in the original paper).

6 Graph orientation

The problem of *graph orientation*, proposed by Horiyama et al. in [9], is as follows: we are given an undirected 3-regular graph, and want to find an orientation of its edges subject to the following constraints on each vertex:

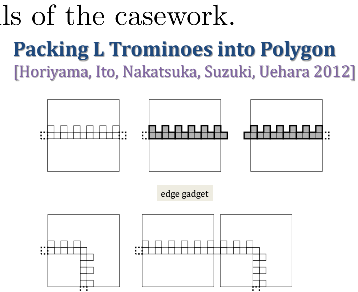
- 1-in-3: exactly 1 incoming and 2 outgoing edges
- 2-in-3: exactly 2 incoming and 1 outgoing edge
- 0-or-3: exactly 3 incoming or 3 outgoing edges

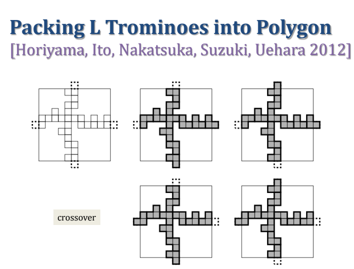
We show this problem is NP-complete by reducing from 1-in-3SAT. For every clause C_i, we build the anticlause and set it to false (by making it a 2-in-3 clause), which enforces 3-regularity of the graph. The truth value of each variable corresponds to whether the edges at the corresponding vertex are directed inwards or outwards (see the right graph in the image below).

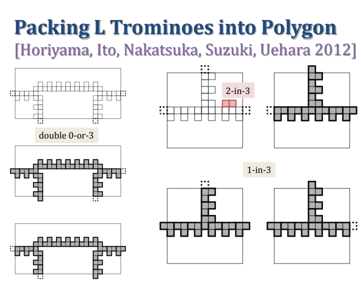


6.1 Packing L-Trominoes into a Polygon

We reduce from the problem of graph orientation. We construct the necessary gadgets (edge and crossover gadgets, 0-or-3 gadget, 1-in-3 and 2-in-3 gadgets) as below. The 0-or-3 gadget comes in the form of a double 0-or-3 gadget, which is the only one required for the reduction from graph orientation (since they are only used in pairs to set the variables). The orientation of the trominoes represents the directedness of the edge. For example, in the edge gadget below, it is easy to check that exactly one of the grid squares outside the large square will be filled. An exact packing will only be possible if we can satisfy the corresponding graph orientation problem. We omit the full details of the casework.

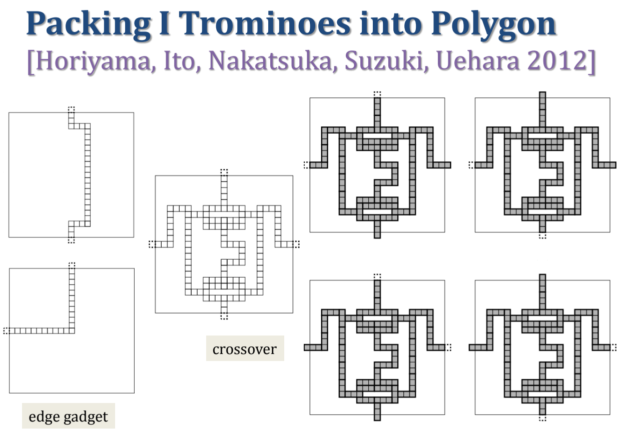


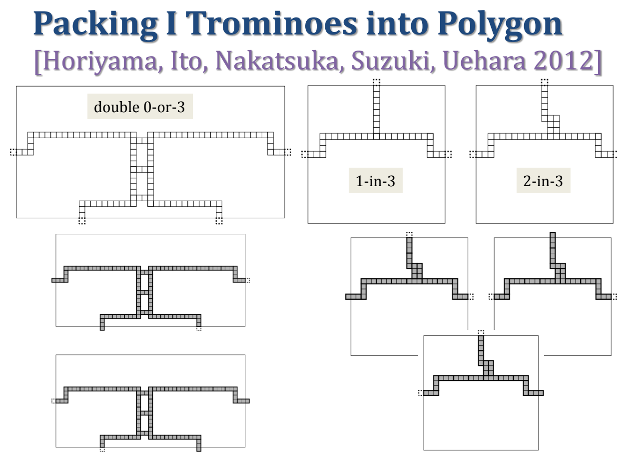




6.2 Packing I-Trominoes into a Polygon

We follow the same techniques as for packing L-Trominoes, and construct the necessary gadgets as below. The bend in the edge gadget is to ensure that only two parities exist (if we just used I shapes in a line, we could have a missing block on one end and a protrusion of 2 blocks on the other end). Other than that, the construction is similar to that of the L-Tromino case.





7 Linear Layout and Crossings

7.1 What is linear layout?

A **linear layout** of a graph G with vertices V and edges E is a bijection of its vertices to the set $\{1, 2, ...|V|\}$. These numbers 1 through $|V|$ correspond to $|V|$ points on a number line in order. The edges of G are then drawn between the points on the line according to the bijection, and some metric is computed upon the resulting figure. Problems in the linear layout family deal with optimizations or decisions about this metric.

7.2 Linear Layout Variants

There are many problems that can be described as linear layout, a collection of which can be found in a survey by Diaz, Petit, and Serna from 2002 [4]. These overarching types of linear layout problems are described here.

Problem	NP-complete	[Díaz, Petit, Serna 2002]
BANDWIDTH	in general	[Papadimitriou 1976]
	for trees with maximum degree 3	[Garey et al. 1978]
	for caterpillars with hair-length ≤ 3	[Monien 1986]
	for caterpillars with ≤ 1 hair per backbone vertex	[Monien 1986]
	for cyclic caterpillars with hair-length 1	[Muradyan 1999]
	for grid graphs and unit disk graphs	[Díaz et al. 2001a]
MINLA	in general	[Garey et al. 1976]
	for bipartite graphs	[Even and Shiloach 1975]
CUTWIDTH	in general	[Gavril 1977]
	for graphs with maximum degree 3	[Makedon et al. 1985]
	for planar graphs with maximum degree 3	[Monien and Sudborough 1988]
	for grid graphs and unit disk graphs	[Díaz et al. 2001a]
MODCUT	for planar graphs with maximum degree 3	[Monien and Sudborough 1988]
VERTSEP	in general	[Lengauer 1981]
	for planar graphs with maximum degree 3	[Monien and Sudborough 1988]
	for chordal graphs	[Gustedt 1993]
	for bipartite graphs	[Goldberg et al. 1995]
	for grid graphs and unit disk graphs	[Díaz et al. 2001a]
SUMCUT	in general	[Díaz et al. 1991]
		[Lin and Yuan 1994b]
		[Golovach 1997]
	for cobipartite graphs	[Yuan et al. 1998]
EDGEBIS	in general	[Garey et al. 1976]
	for graphs with maximum degree 3	[MacGregor 1978]
	for graphs with maximum degree bounded	[MacGregor 1978]
	for d-regular graphs	[Bui et al. 1987]

- *Bandwidth* - Let the points be laid out on a number line so that the distance between consecutive points is a unit length. Minimize the maximum length of any edge between two points. This problem is motivated by the concept of bandwidth in linear algebra. In linear algebra, matrices whose nonzero values are all in a thin band around the main diagonal are much easier to manipulate and, if they represent a system of linear equations, easier to solve. Permuting the columns and rows of an adjacency matrix is analogous to permuting the points on a linear layout. Bandwidth problems are hard even for very constrained graphs, such as trees of maximum degree 3 and caterpillars.

- *MinLA* - Short for minimum linear arrangement, the goal in this problem is to minimize the *sum* of the edge lengths (as opposed to the maximum edge length, as in bandwidth). MinLA problems are motivated by VLSI chip design.

- *Cut width* - Let the *size* of a cut in a linear layout be the number of edges with one endpoint at i or less and the other endpoint at $i+1$ or more, for some i. Minimize the maximum size of any cut. Note that attempting to minimize the sum of the cuts is identical to minLA.

- *Vertex separation* - This problem is similar to cut width, except that we consider the *vertex separation* of a cut, which is the number of vertices that have at least one edge crossing the cut. We aim to minimize the maximum vertex separation of any cut.

- *Sum cut* - This problem uses the same setup as vertex separation. The only difference is that we aim to minimize the sum of the cuts instead.

- *Edge bisection* - Minimize the number of edges that cross the middle cut.

- *Vertex bisection* - Minimize the number of vertices in the left half with edges to the right half.

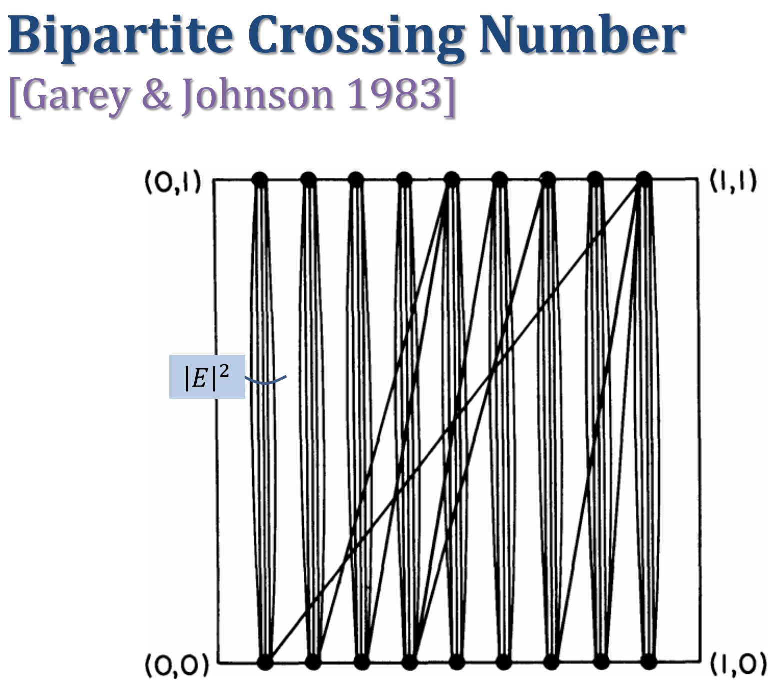
- *Betweenness* - This problem is not posed in a graph-based way. Instead, you are given a list of rules of the form "y is between x and z," meaning that either "$x < y < z$" or "$z < y < x$". The goal is to impose a total ordering (like a linear layout of all the symbols) that satisfies all the rules. This problem was originally posed by Opatrny in 1979 [11].

7.3 MinLA and Bipartite Crossing Number

We now turn to a different problem, that of determining the crossing number of a graph. The **crossing number** of a graph is a metric somewhat related to planarity. It is defined as the minimum number of crossings between edges (excluding edges that meet at vertices) over all drawings of the graph in 2 dimensions. We will show this problem is NP-Hard using a two-step reduction from the MinLA variant of linear layout. The entire reduction that follows is due to Garey and Johnson [6].

First, we introduce an intermediate problem, bipartite crossing number. In this problem, we are given a bipartite graph, and we must arrange its vertices on two parallel straight rails (one rail per part) in order to minimize the number of crossings of edges. The edges pass outside the area between the two rails. We reduce from MinLA as follows.

Given a MinLA problem G with vertices $V = \{v_0, \ldots v_m\}$ and edges E, we first duplicate the vertices. Create a v_i' for each v_i. Now, we modify the edges so that they run between a point and its duplicate version instead. If an edge originally ran between v_i and v_j for $i < j$, modify it to instead connect v_i and v_j'. This transforms the MinLA graph into a bipartite graph. Lastly, we attach large bundles of B parallel edges between v_i and v_i' for each i (B is some number larger than $(|V| + |E|)^2$). This prevents rearrangement of the vertices between rails: the order of the original vertices must be the same as the order of the new vertices (if two vertices were rearranged, we would get B^2 crossings from the bundles alone).

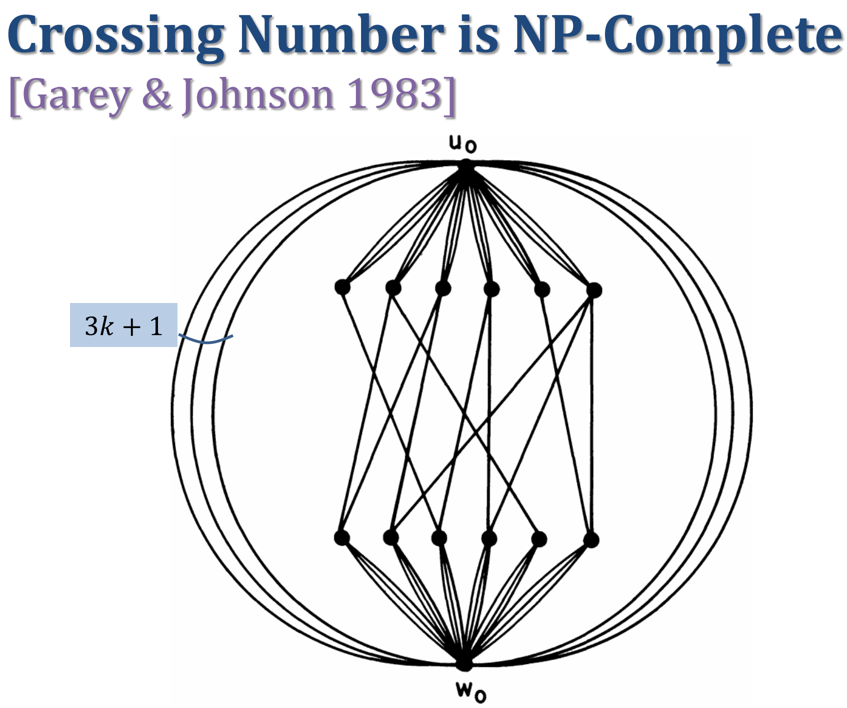


Now consider the number of crossings of an original edge. The number of bundles it crosses is exactly one less than the length of the edge in a linear layout with the same order on the vertices. The crossings between original edges are negligible in comparison, since crossing one bundle adds B crossings but the original edges can only cross each other $|E|^2$ times. Since each bundle is the same size and the number of edges is constant, minimizing the crossing number of this bipartite graph also solves the original MinLA problem.

7.4 Bipartite Crossing Number and General Crossing Number

Finally, we reduce from the intermediate problem of bipartite crossing number to the problem of crossing number in the case of general graphs. This reduction was introduced in the same paper as the one in the previous section [6].

Clearly, a bipartite graph is an example of a general graph, so all we have to do is impose some additional structure on the bipartite graph to mimic the 'two rails' condition from the bipartite crossing number problem. To do so, add two 'bounding' vertices X and Y (the top and bottom vertices in the following diagram). Connect X to the first part of the bipartite graph using large bundles, and connect Y to the other part. Then connect X and Y to each other using large bundles twice. Large bundles should be significantly larger than B (B^4 is probably safe, but smaller numbers may work as well). Then, the drawing of the bipartite graph is forced as shown.

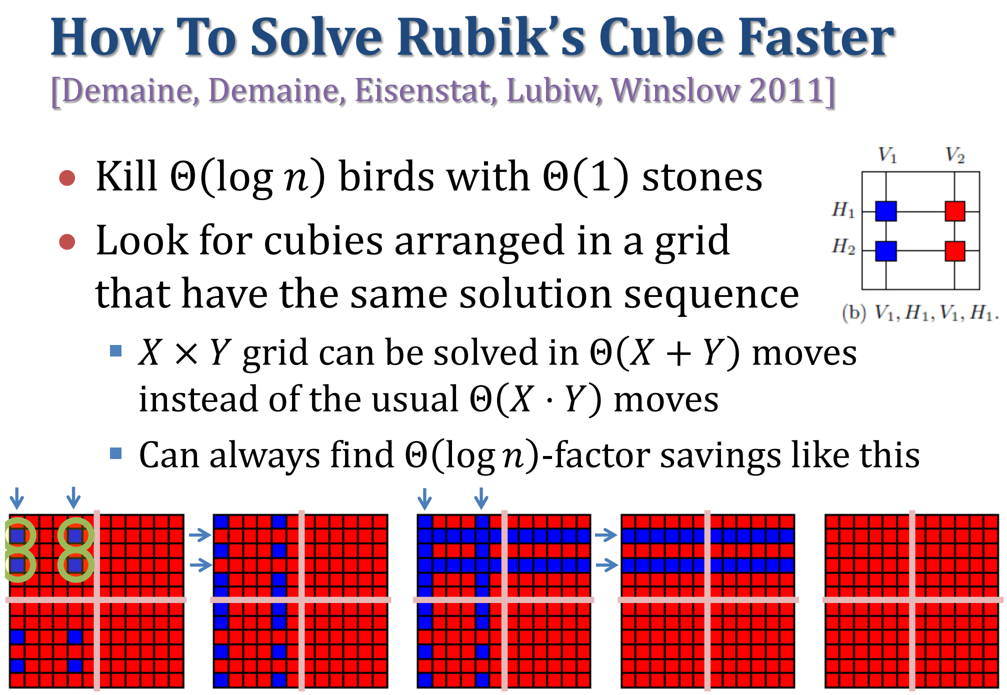


Having reduced from MinLA, which is NP-hard, to crossing number, we see that finding the crossing number of a graph in general must also be NP-hard.

8 Solving Rubik's Cubes

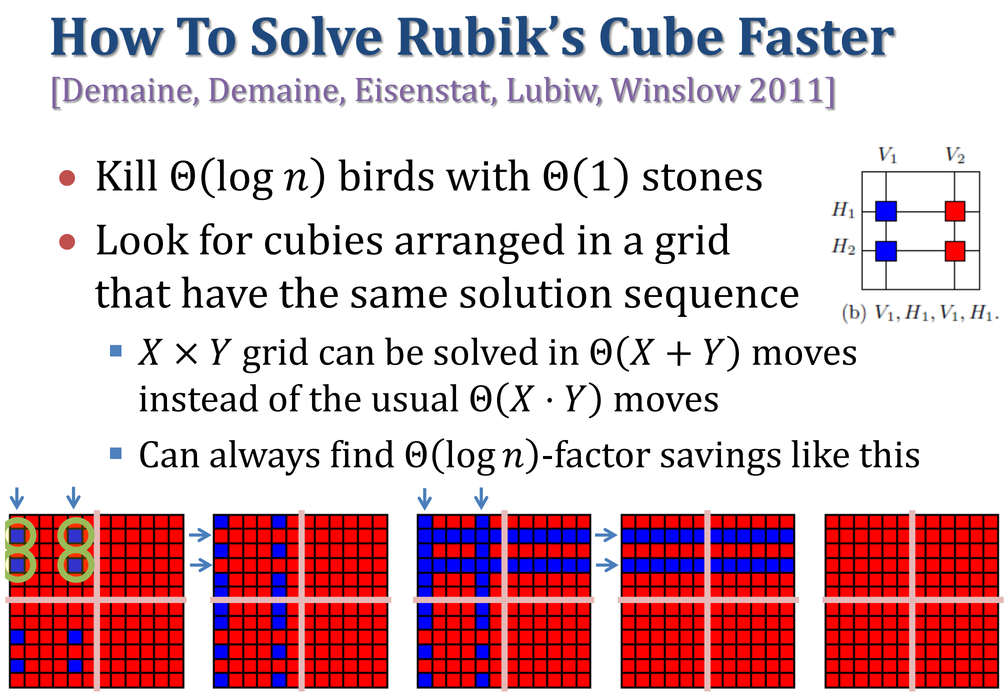
We briefly mention two results regarding Rubik's Cubes here.

8.1 Faster solving



Demaine et al. determined an $O(\frac{n^2}{\log n})$ algorithm for solving a Rubik's Square, which is a n by n by 1 variant of a Rubik's Cube [2]. On a high level, this is done by identifying $\Theta(\log n)$ **'cubies'** (single faces) that can be solved with the same solution sequence. In the above figure, the four circled cubies can be solved simultaneously by a vertical flip on those two columns, followed by a horizontal flip on those two rows, then another vertical and horizontal flip. Demaine et al. showed that they can always find such a set of cubies to solve as a batch, thus providing a $\Theta(\log n)$ factor of savings.

8.2 Hardness of optimal solving



In the same paper [2], Demaine et al. also showed that if you only care about a subset of the stickers on a Rubik's Cube, solving it is NP-Hard. The reduction is from betwenness (recall that betweenness is a problem related to linear layout in which you must order some objects according to rules of the form "y is between x and z). In the above figure, the first time column x_2 is flipped must be between the first flips of x_1 and x_3. The details of the reduction are not reproduced here.

It remains an open problem whether finding the optimal solution is hard if all cubies are important (this question was posed by Andy Drucker and Jeff Erickson in 2010 on the StackExchange forum at `http://cstheory.stackexchange.com/questions/783`).

References

[1] R. L. Brooks. On colouring the nodes of a network. *Mathematical Proceedings of the Cambridge Philosophical Society*, 37:194–197, 4 1941.

[2] Erik D. Demaine, Martin L. Demaine, Sarah Eisenstat, Anna Lubiw, and Andrew Winslow. Algorithms for solving rubik's cubes. In *Algorithms - ESA 2011 - 19th Annual European Symposium, Saarbrücken, Germany, September 5-9, 2011. Proceedings*, pages 689–700, 2011.

[3] Erik D. Demaine, Martin L. Demaine, Michael Hoffmann, and Joseph O'Rourke. Pushing blocks is hard. *Comput. Geom.*, 26(1):21–36, 2003.

[4] Josep Díaz, Jordi Petit, and Maria Serna. A survey of graph layout problems. *ACM Comput. Surv.*, 34(3):313–356, September 2002.

[5] Erich Friedman. Pushing blocks in gravity is np-hard. *Unpublished manuscript, March*, 2002.

[6] M. Garey and D. Johnson. Crossing number is np-complete. *SIAM Journal on Algebraic Discrete Methods*, 4(3):312–316, 1983.

[7] M. R. Garey and David S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman, 1979.

[8] M. R. Garey, David S. Johnson, and Larry J. Stockmeyer. Some simplified np-complete graph problems. *Theor. Comput. Sci.*, 1(3):237–267, 1976.

[9] Takashi Horiyama, Takehiro Ito, Keita Nakatsuka, Akira Suzuki, and Ryuhei Uehara. Packing trominoes is np-complete, #p-complete and asp-complete. In *Proceedings of the 24th Canadian Conference on Computational Geometry, CCCG 2012, Charlottetown, Prince Edward Island, Canada, August 8-10, 2012*, pages 211–216, 2012.

[10] David Lichtenstein. Planar formulae and their uses. *SIAM J. Comput.*, 11(2):329–343, 1982.

[11] Jaroslav Opatrny. Total ordering problem. *SIAM J. Comput.*, 8(1):111–114, 1979.