6.890 Lecture 17 Scribe Notes

1 Overview

Today we will talk about constraint logic and its uses in Modeling Computation as Games. This was part of Bob Hearn's thesis at MIT [2]. In particular, we will consider Nondeterministic Constraint Logic (NCL) problems and use them to show that problems such as Rush Hour and Sliding Blocks are PSPACE-Complete, as we noted in lecture 1.

We will first define the problem and motivate why we can think of it as a model of computation. Then, we will construct a series of logical gadgets that will be useful for reductions. Afterwards, we will show certain decision problems stemming from NCL are PSPACE-Complete. Finally, we will use these results to show that many puzzles and problems are PSPACE-Complete.

2 Constraint Graphs

We will introduce Constraint Graphs as computation models. In these models, we will think of the whole (undirected) graph as a machine. The graph has two types of edges: red edges with weight 1 and blue edges with weight 2. A configuration consists of a machine equipped with an orientation of the edges. We say that an orientation is valid if it satisfies the inflow constraint: each vertex must have an incoming weight of at least 2. A move consists of reversing an edge in such a way that the resulting configuration is still valid.

For today's lecture, we will discuss about the following two decision problems.

- Given a constraint graph, can I reverse a specified edge by a sequence of valid moves? Note that this could be done by a sequence of valid moves so long as the last valid move reverses the desired edge.
- Given a constraint graph, can I reach a desired configuration by a sequence of valid moves?

The punchline is that both of these problems are PSPACE-Complete for 3-regular or max-degree 3 graphs.

Another interesting question is, given an undirected graph, is there an orientation of the edges that satisfies the inflow constraints? This is the Constraint Graph Satisfaction problem and we will show later it is NP-Complete [1].

2.1 Logical Gadgets

In fact, we will show these problems are PSPACE-Complete even for just 2 special vertex types. See figure 1 for the construction.

- An AND vertex is a vertex such that it has two incident red edges (inputs) and one blue incident edge (output).
- An OR vertex is a vertex such that it has three incident blue edges (two inputs, one output).

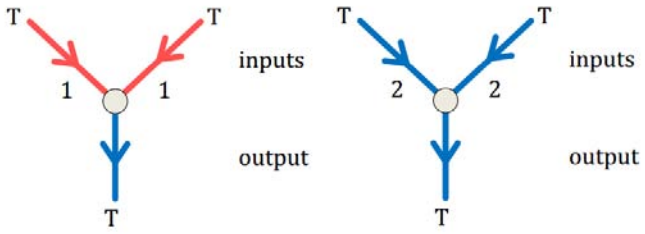

Figure 1: And Vertex and Or Vertex

We say an output edge can activate if it can be directed out. We say an input edge is active if it can directed in. This redefines AND and OR vertices. In particular, AND vertices are those such that the output edge can activate only if the inputs are active. Similarly, an OR vertex is one such that the output edge can activate if either of the inputs are active. This almost matches our notion of AND and OR. The one caveat is that there may be a delay between when an output edge can activate and effectively becomes active. This can lead to some cases that differ with our normal notion of an OR gate.

We can also build a SPLIT gadget (see figure 2) which takes two red edges as outputs and one blue edge as input. The outputs can be active only if the input is active. This is an alternative view of the AND vertex.

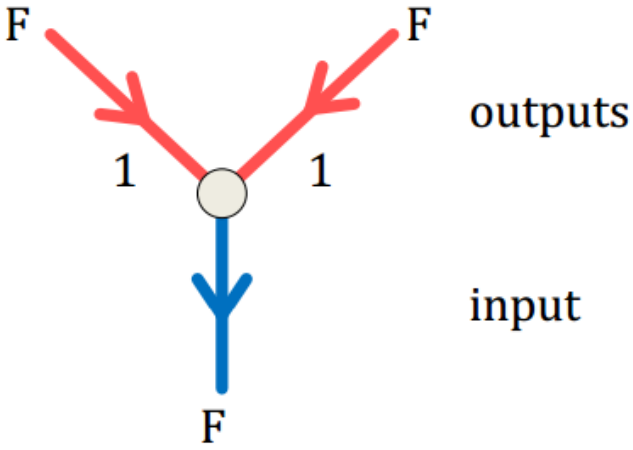

Figure 2: Split Vertex

It is important to note at this point that we can not have a NOT gate on this construction. The idea of a NOT gate is to have a vertex such that the output can be active only if input is not active. But such a vertex would violate the inflow constraint.

2.2 Some more useful gadgets

The following gadgets will be useful for our reductions.

- A CHOICE vertex will be represented as a node with one red input edge and two red output edges. The idea is that if the input is active, only one of the two outputs can be active, hence the choice. An equivalent construction would use one SPLIT as input and two ANDs as outputs. For the construction, see figure 3.
- A RED-BLUE conversion gadget is used to turn the output of an AND or an OR vertex into the input for an AND or CHOICE vertex. This changes the color of the edge from blue to red. For the construction see figure 4.
- A Wire Terminator is a gadget that takes a degree one vertex and turns it into a degree 3 vertex. Special cases arise depending on the color and desired orientation of the wire. For the construction see figure 5.

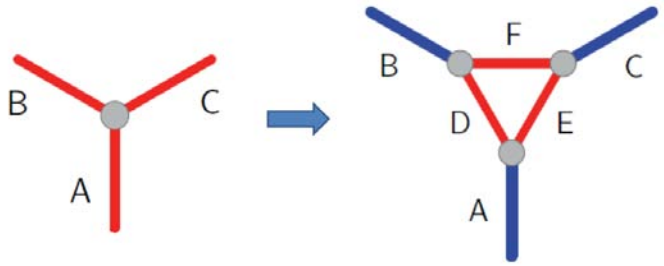

Figure 3: Choice Vertex

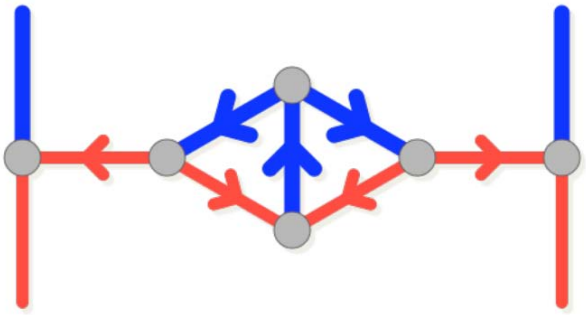

Figure 4: Red-Blue Conversion

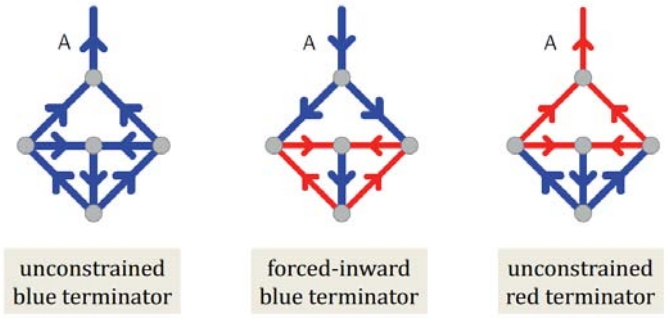

Figure 5: Wire Terminators

These, in combination with the ANDs and ORs, allow us to simulate any CNF formula. This immediately implies that the Constraint Graph Satisfaction problem is NP-Complete by a reduction from 3SAT. The reduction gadget is shown in figure 6.

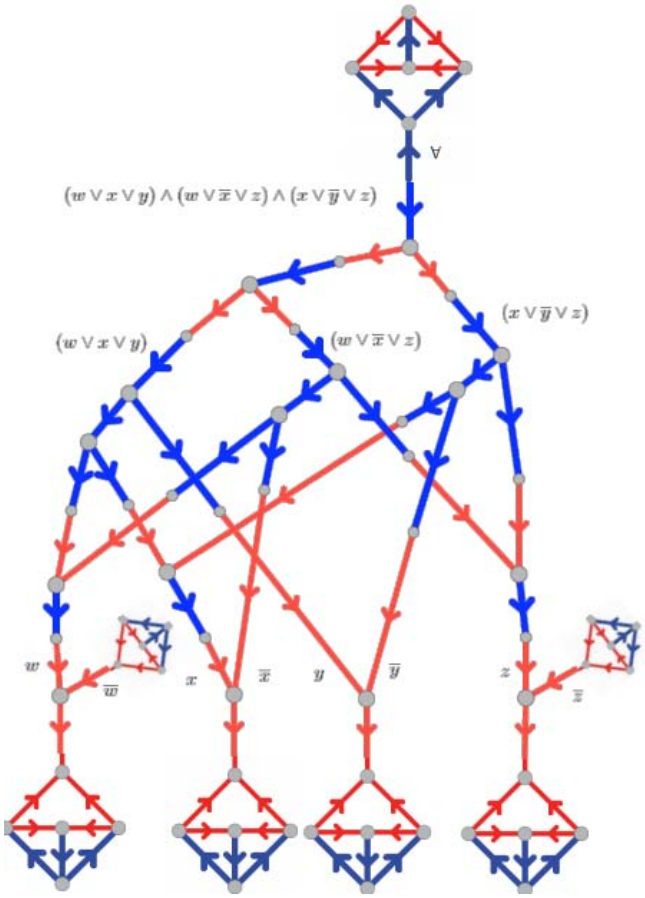

Figure 6: Constraint Graph Satisfaction is NP-Complete

3 Showing that NCL is PSPACE-Complete

We will reduce from QSAT. We will have some quantifier gadgets for universal and existential clauses. The reduction gadget is shown in figure 7.

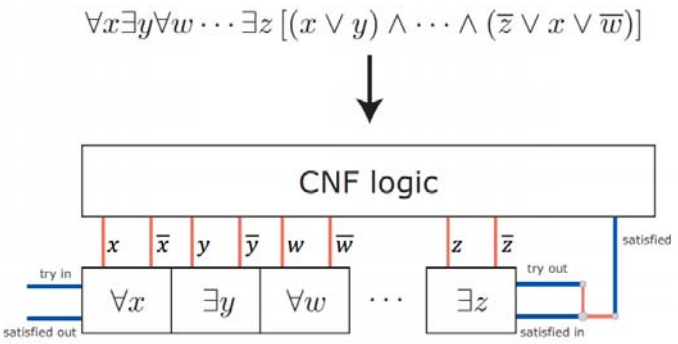

Figure 7: NCL is PSPACE-Complete

3.1 Existential and Universal Gadgets

The basic idea of these gadgets is as follows. Each receives feedback from the previous one on what to do, tries it out and based on the result it gets, transmits a message to the next gadget.

For the existential quantifiers (figure 8), we only require that there exists some assignment that gives a satisfying orientation. Once we found such a value (if it exists), a truth value will be propagated down the 'try out' path. Otherwise, a false value will be propagated down. This will ultimately force the satisfied in to be false and as a consequence satisfied out will also be false.

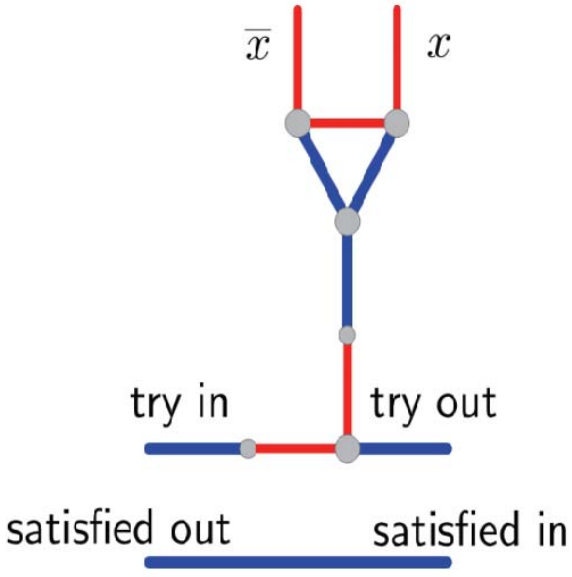

Figure 8: Existential Quantifier

On the other hand, the universal quantifier (figure 9) will only return true when all possible assignments give a satisfying orientation. Notice that this gadget is connected to the satisfied out wire. This is done in case we find an unsatisfiable clause. If such a universal clause exists, we pass along a false value on the satisfied out channel. If for all assignments there is an orientation, a truth value will be passed down through try out to the next gadget.

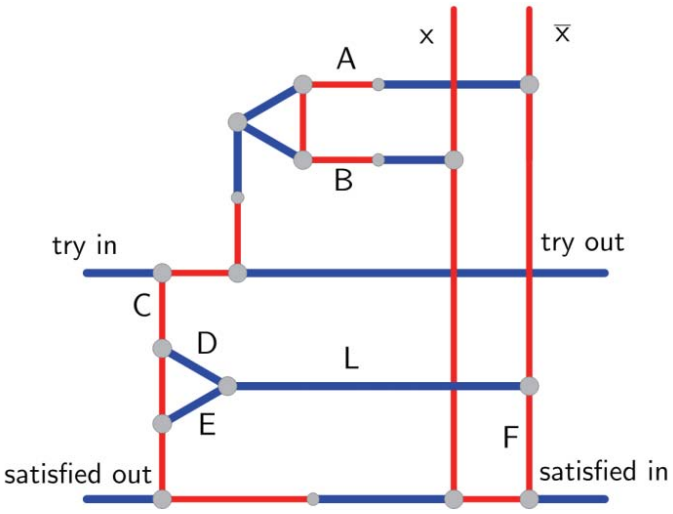

Figure 9: Universal Quantifier

3.2 Supplementary Gadgets

The LATCH gadget (see figure 10) consists of three vertices, one of which is incident to three blue edges and the other two are incident to two red and one blue edge. If the gadget is locked, then only one of the two outgoing red edges can be an output. If the gadget is unlocked, both red edges can be outgoing. (see picture).

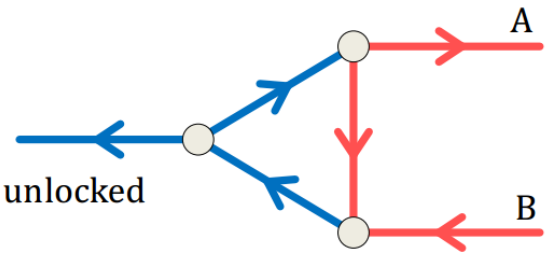

Figure 10: Latch Gadget

Notice that the satisfied out edge will be reversible if and only if the formula has a satisfying assignment. Therefore the problem of deciding whether or not an edge can be flipped is PSPACE-complete by a reduction from QSAT.

What about going from one configuration to another? We can do this by adding a latch at the

beginning of the gadget, and asking if we can open the latch. This will be possible if and only if the configuration is reachable. Therefore this problem is also PSPACE-Complete.

3.3 About Planarity

But there's more! We can also show that NCL is PSPACE-Complete if the constraint graph is planar. All we need is a crossover gadget, pictured below.

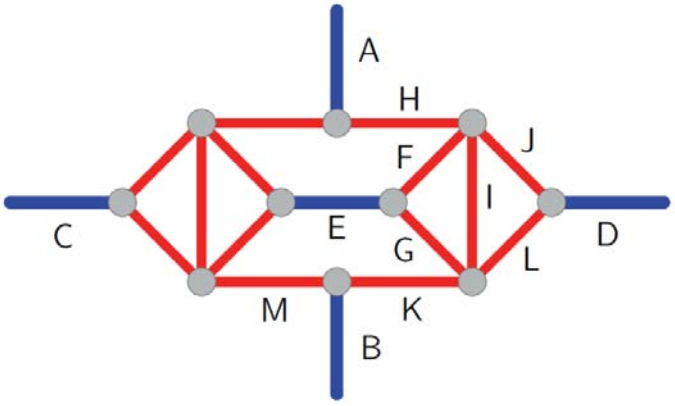

Figure 11: Blue-Blue Crossover

Figure 11 shows a blue-blue crossover. There are some vertices with degree 4, but there's a special gadget to turns such a vertex into one with degree 3. Refer to figure 12 for that gadget.

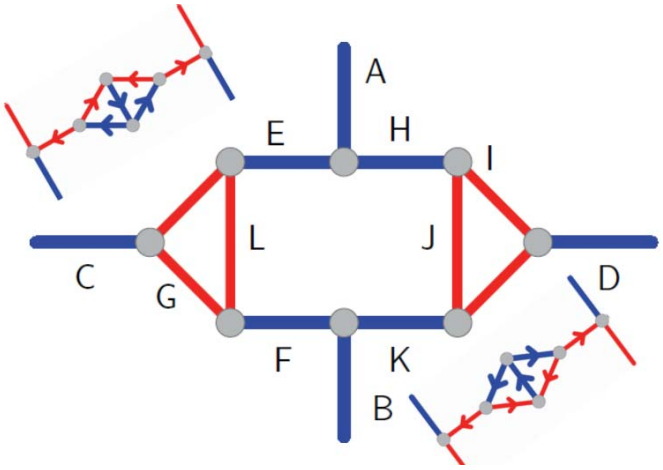

Figure 12: Getting Rid of Vertices Incident With Four Edges

This gadget also works for a red-red crossover. For other crossovers, we can use a red-blue conversion first and then use gadget. This shows that the planar version of the problem is also PSPACE-Complete.

3.4 About Grid Graphs

We can also consider the problem of Grid Constraint Graphs. Recall that we can always turn a planar graph into a grid graph. This construction turns edges of the graph into paths on the grid. Therefore, we need straight gadgets, turn gadgets to simulate paths as well as filler and OR gadgets. See figure 13 for the described gadgets.

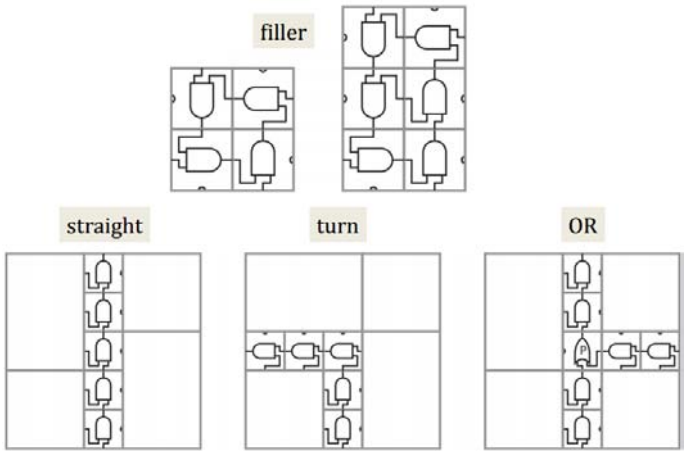

Figure 13: Grid Graph additional gadgets: fillers, straights, turns and ORs.

3.5 Protector OR

Protector OR. We want to have at most ≤ 1 input active at any time. We can get normal ORs from protected ORs. The goal is to get exactly one input going into the OR gadget.

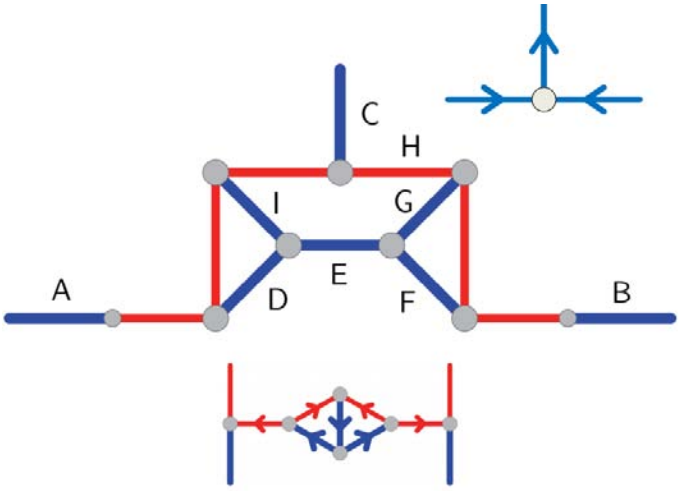

Figure 14: From OR to Protector OR

4 Reconfiguration 3-SAT [3]

Given two satisfying assignments, is there a sequence of valid moves that takes us from one assignment to another? In this context, a move is flipping a variable's value. This can be shown to be PSPACE-Complete via an easy reduction from the configuration reachability NCL problem.

In this construction, each edge is a variable. In particular, an incoming edge is true if it is active and an outgoing edge is true if it is active. So an OR clause looks like either x in, or y in, or z in. An AND clause looks like two red edges and 1 blue edge, that says x out implies y in and x out implies z in. Therefore, we can simulate any formula by taking the AND of all the clauses in the original NCL graph.

5 Back to Square One

Now we will use the machinery we've just developed to show that a bunch of puzzles are PSPACE-Complete by reductions from NCL Configuration Reachability.

- Sliding-Block Puzzles [4]: Blocks are rectangles. We can slide them along non-colliding paths. The goal is to move a certain block to a certain position. This remains PSPACE-Complete even if the rectangles are 1×2.
- Sliding Tokens: Given two independent sets of a graph, we can think of the selected vertices as tokens. Is there a valid way to slides the tokens to get from one set to the other?
- Rush-Hour [4], [5]: This goes way back to lecture 1, except now we have the tools to show it. Suppose we are given a rush-hour configuration and we want to find out whether or not we can move a particular block to a particular location at the edge of the grid. This problems also remains PSPACE-Complete even if the blocks are 1×2 [6]. An open question however is to classify Rush Hour when the blocks are 1×1. It is not known whether it is in P or PSPACE-Complete [6].
- Triangle Rush Hour [1]: still PSPACE-complete. It's like Rush-Hour but with triangular shapes.
- Hinged Dissection [1]: Given a polygon and a hinged dissection, can we reach another given configuration by a continuous movement of the hinged parts?
- Sokoban [4]: Suppose we are given a configuration of 1×1 blocks and a set of target positions. We designate one block as a pusher. A move consists of moving the pusher a single unit either horizontally or vertically. There are some additional rules. The goal is to make a sequence of moves such that there is a block on each target position. [4]
- Rolling Block Mazes [7]: can we roll a $k \times m \times n$ block through a maze from it's starting position to a specified ending position. The most common version of this is $2 \times 1 \times 1$, with the specified position being the block 'standing up'. Plank puzzles, which are similar, have also been shown PSPACE-Complete.
- Dynamic Map Labeling [8]: given a static map, is there a smooth dynamic labeling when the points move, when points are added or removed, or when the user pans, rotates, and/or zooms their view of the points.

References

[1] Robert A. Hearn, "Games, Puzzles, and Computation". Ph.D. thesis, Massachusetts Institute of Technology, 2006.

[2] Erik D. Demaine and Robert A. Hearn, "Constraint Logic: A Uniform Framework for Modeling Computation as Games", in *Proceedings of the 23rd Annual IEEE Conference on Computational Complexity (Complexity 2008)*, College Park, Maryland, June 23 - 26, 2008, pages 149 - 162.

[3] Gopalan, P., Kolaitis, P. G., Maneva, E. N., Papadimitriou, C.H., "The connectivity of Boolean Satisfiability: computational and structural dichotomies.", in *SIAM Journal on Computation (SICOMP) 38(6), 2009.*

[4] Robert A. Hearn and Erik D. Demaine, PSPACE-Completeness of Sliding-Block Puzzles and Other Problems through the Nondeterministic Constraint Logic Model of Computation, Theoretical Computer Science, volume 343, number 1-2, October 2005, pages 72-96. Special issue "Game Theory Meets Theoretical Computer Science".

[5] Gary William Flake and Eric B. Baum. "Rush Hour is PSPACE-complete, or Why you should generously tip parking lot attendants. *Theoretical Computer Science*, 270(12):895911, January 2002.

[6] John Tromp, Rudi Cilibersi. "Limits of Rush Hour Logic Complexity". 2006

[7] M. Holzer, S. Jakobi. "On the Complexity of Rolling Block and Alice Mazes" in *E. Kranakis, D. Krizanc, F. Luccio (eds.): Proceedings 6th International Conference on Fun with Algorithms (FUN 2012)*, volume 7288 of LNCS, pages 210-222, Venice, Italy, June 2012. Springer.

[8] Buchin, K., Gerrits, D.H.P. (2013). "Dynamic point labeling is strongly PSPACE-complete", in *L. Cai, S.-W. Cheng, T.-W. Lam (Eds.), Conference Paper : Algorithms and Computation (24th International Symposium, ISAAC 2013, Hong Kong, December 16-18, 2013. Proceedings)*, (Lecture Notes in Computer Science, 8283, pp. 262-272). Berlin: Springer.

6.890 Algorithmic Lower Bounds: Fun with Hardness Proofs

Fall 2014