Notes on chapter 12

Instructor: Mohammad T. Hajiaghayi
Scribe: Benjamin Sela

1 General comments

I think I found the gadgets to be pretty confusing. When looking at the figures I had no idea which vertices I should look at, and having an example that is simpler than Figure 12.6 might have been helpful as well. When presented with a gadget, there are a lot of vertices and it is hard to understand which ones were added by the gadget, and which ones correspond to something from the original problem you are modifying.

2 misc typos

I did not find any.

3 Comments on section 12.2.1

1. **Explanation of input/output edges and edge orientation** I found the relationship between input/output edges and the concept of edge direction to be a bit confusing as it was first presented. For instance, right below figure 12.1 (on page 304), it says "We say an output edge can activate if it can be directed out. We say an input edge is active if it can be directed in". When reading this I had no idea what it means to say that an edge CAN be directed in some direction. Instead saying something like "We say an output edge can activate if it can be directed out subject to the constraint that the machine is valid"

2. **Definition of input/output edges:** I found the concept of an edge being an input or output edge very confusing because as far as I could tell this was not built into the definition of a machine. My understanding is the notion of an edge being an output or an input is based on how we intend for a valid configuration to orient that edge, based on the problem we are modeling (in this case SAT problems). If this is correct, then the terms "output" and "input" are only defined in the context of designing gadgets or simulating some other problem. On the other hand, edges are

not inputs or outputs if we just look at an arbitrary CGS problem. The discussion under Figure 12.1 makes it seem like an edge being an input or output is built into the specification of the machine which is not the case.

Based on the points above I suggest adding something like the following into section 12.2.1.

We now construct gadgets which simulate the behavior of logical gates. In particular, we wish to define machines which have a valid configuration if and only if the corresponding logical gate outputs TRUE. To make this idea precise, we need to define what it means for an edge to be an input/output, and we also need a notion of which boolean value an edge is assigned.

DEFINITION [**l**ogical interpretation of CGS]

Let g be a vertex in a machine, and assume we have labeled each edge incident to g as either an input or output with respect to g (note the edges are still not directed yet, we are simply stating the direction we hope they will have in the future).

1. We think of an edge as carrying the value TRUE iff a valid configuration for G orients the edge in the same direction as the input/output assignment.
2. An output (input) edge oriented away (towards) g is interpreted as the corresponding gate having output (input) value TRUE.
3. The reverse situation corresponds to an output or input carrying the value FALSE.

END DEF

4 Comments on section 12.2.2

1. **Figure 12.6:** If possible I would render figure 12.6 at a higher resolution. There is a lot going on and the text looked pretty fuzzy.
2. I found the explanation of the gadgets on page 305 confusing and I found that I was not sure what to look for in the corresponding figures. I also think it might be good to additionally have a simpler example. For instance, including an example such as $\varphi = (x \vee y) \wedge (y \vee z)$ to help motivate the logical gadget section would have helped me.
3. **Red-Blue Conversion:** It took me a while to figure out which edges/vertices in Figure 12.4 corresponded to the conversion and which ones were just part of the gadget. It is pretty obvious in hindsight but it was not clear at first. I think something along the lines of the following would have helped me follow it better.
   (a) Each instance of the construction below will allow for the conversion of two edges. The gadget itself is in the center and the conversion

is happening at vertices c_1 and c_2. We view vertices g_1 and g_2 as gates, and the blue edges touching them are the outputs of those gates. This gadget insures

i. that the logical value on each side of c_i will be the same (if the blue edge is pointing up (FALSE) then the red edge must also be pointing up (FALSE)).

ii. that both logical values are possible. The sub graph in the center ensures that there will always be a valid configuration in which c_1 and c_2 have an incoming red edge. This means that as long as the orientation of the remaining two edges are opposite (with respect to c_i), either set of orientations is valid. This means it is possible for gates g_1 and g_2 to output either TRUE or FALSE, as would be the case in the corresponding formula.

5 Comments on 12.2.3

1. **Theorem 12.2.4 and Crossover Gadgets:** I think it would be good to briefly describe in 1-2 sentences what a crossover gadget does. Even just having a before/after figure that shows how the gadget removes edge intersections, thus making the graph planar. Also, I did not really understand figure 12.8. I trust the authors that there is a way to get rid of degree 4 vertices, but I have no idea what is going on with the red-blue conversion gadgets floating in the corners. I am not sure if comparing the edge labels between figures 12.7 and 12.8 is supposed to make that clear but I was not able to figure it out.

2. **Theorem 12.2.4** When reading the proof it was a little hard to understand what we were trying to do at first, and I would reword the theorem proof slightly as follows.

   BEGIN PROOF

   (a) In order to model a 3SAT formula using a planar graph, we will take the graph construction used in Theorem 12.2.3, and then introduce a gadget which eliminates intersections between edges. We call this gadget a crossover gadget (See Figure 12.7).

   (b) The previous bullet point solves the first half of the theorem. The original constraint graph construction from theorem 12.2.3 was of degree 3, and the crossover gadgets just used introduced several degree 4 vertices. We use the following additional gadget (see Figure 12.8) to reduce the degree back down to 3.

   END PROOF

6 Related complexity problems

1. **Hardness of NCL on bounded bandwidth graphs:**

   DEFINITION

   Let $G = (V, E)$ be a graph and define a one-to-one correspondence $f : V \mapsto \{1, ..., |V|\}$. The bandwidth of G is the minimum over all such mappings f of $\max_{(u,v) \in E} |f(u) - f(v)|$.

   END DEFINITION

   Van Der Zanden [1] showed that NCL remains PSPACE complete on graphs with bounded bandwidth.

2. **Fixed Parameter Tractability of NCL:** View the number of weight one edges, the number of weight two edges, and the number of AND/OR vertices (in a AND/OR constraint graph) as parameters to an NCL instance. Hatanaka et. al. [2] showed that NCL is fixed parameter tractable with respect to each of these parameters.

3. **Snowman is PSPACE complete:** Snowman is a SOKOBAN-like puzzle game released in 2015. He et. al. [3] showed that snowman is PSPACE complete.

4. **Reachability (Chess):**Given an $n \times n$ Chess position, is it possible to reach that position from a starting position? Brunner et al. [7] showed that this problem is PSPACE complete by a reduction from Subway Shuffle.

5. **Helpmate (Chess):** Given an $n \times n$ Chess position, is it possible to reach a position in which the black king is checkmated? Brunner et al. [7] showed that this problem is PSPACE complete by a reduction from Subway Shuffle.

6. **Motion Planning:** There are many games that fall into this category, most famously SOKOBAN. There are a handful of block pulling games in which an agent must achieve a goal by pulling (instead of pushing) blocks around. Any et. al. [4] showed that many variants of block pulling games are PSPACE complete via reductions from NCL.

7. **2048 ($n \times n$ variant) is PSPACE Hard:** 2048 is a mobile game that was popular around 2014. Rahul [8] showed this result by a reduction from planar NCL

8. **Reconfiguration of Satisfying assignments:** Given two two satisfying assignments to a planar monotone instan ce of Not-All-Equal 3-SAT, can one assignment be transformed into the other by a sequence of single variable flip, while ensuring that each intermediate assignment satisfies the formula? Cardinal et al. showed this problem to be PSPACE complete by a reduction from NCL.

9. **Snake-Like Robots:** Given a snake-like robot with an initial position and final position in a graph, can the robot reach the final position from the initial position without intersecting itself? Gupta et. al. [5] showed that this problem is FPT.

10. **Reconfiguration of Subset Sums:** Given two subsets S of integers with the same sum, can one subset be transformed into the other by adding or removing at most three elements of S at a time, such that the intermediate subsets always have the same sum as each other. Cardinal et al. [6] showed this problem was PSPACE complete via reductions from NCL.

References

[1] Van Der Zanden, Tom C. "Parameterized Complexity of Graph Constraint Logic." Application/pdf, 2015, 12 pages. https://doi.org/10.4230/LIPICS.IPEC.2015.282.

[2] Hatanaka, Tatsuhiko, Felix Hommelsheim, Takehiro Ito, Yusuke Kobayashi, Moritz Muhlenthaler, and Akira Suzuki. "Fixed-Parameter Algorithms for Graph Constraint Logic," n.d., 22.

[3] He, Weihua, Ziwen Liu, and Chao Yang. "Snowman Is PSPACE-Complete." Theoretical Computer Science 677 (2017): 31–40. https://doi.org/10.1016/j.tcs.2017.03.011.

[4] Ani, Joshua, Sualeh Asif, Erik D. Demaine, Yevhenii Diomidov, Dylan Hendrickson, Jayson Lynch, Sarah Scheffler, and Adam Suhl. "PSPACE-Completeness of Pulling Blocks to Reach a Goal." Journal of Information Processing 28, no. 0 (2020): 929–41. https://doi.org/10.2197/ipsjjip.28.929.

[5] Gupta, Siddharth, Guy Sa'ar, and Meirav Zehavi. "The Parameterized Complexity of Motion Planning for Snake-Like Robots," n.d., 33.

[6] Cardinal, Jean, Erik D. Demaine, David Eppstein, Robert A. Hearn, and Andrew Winslow. "Reconfiguration of Satisfying Assignments and Subset Sums: Easy to Find, Hard to Connect." Theoretical Computer Science 806 (February 2020): 332–43. https://doi.org/10.1016/j.tcs.2019.05.028.

[7] Brunner, Josh, Erik D. Demaine, Dylan Hendrickson, and Julian Wellman. "Complexity of Retrograde and Helpmate Chess Problems: Even Cooperative Chess Is Hard." ArXiv:2010.09271 [Cs], October 19, 2020. http://arxiv.org/abs/2010.09271.

[8] Mehta, Rahul. "2048 IS (PSPACE) HARD, BUT SOMETIMES EASY," n.d., 13.