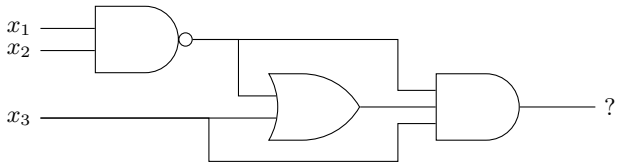
Class 4 Scribe Notes	Notetakers: Jason Ku, Yun William Yu
6.890	
Instructor: Erik Demaine	*2014-09-16*

1 Welcome

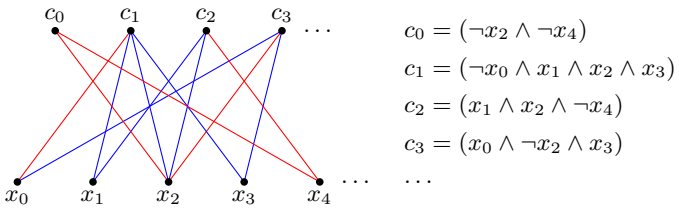
Quiet on set. Today we start a series of lectures on Satisfiability. We saw little bit about this in the first lecture, but today we'll do it right, using SAT to prove NP-hardness. SAT is really the most common problem used to prove NP-hardness, so this is the moment you've all been waiting for, where we go through a whole bunch of SAT variations and classify them as hard or easy.

2 Some NP-hard variants

- **SAT = Satisfiability:** [Cook 1971; Levin 1973]
  - The most important NP-complete (logic) problem family
  - given a Boolean formula (AND, OR, NOT) over n variables $x_1, x_2, ..., x_n$.
  - can you set x_i's to make the variable true.
- **Circuit SAT:**
  - formula expressed as circuit (DAG) of gates (allowing reuse)
  - NP complete when acyclic
  - This is NP because smart guessing works to finding a path through the circuit.
  - Example of a circuit gate:



- **CNF (Conjunctive Normal Form) SAT:**
  - Form:
    - formula = AND ("$\vee$") of clauses
    - clause = OR ("$\wedge$") of clauses
    - literal = x_i / $\neg x_i$ ("NOT x_i")
  - (NB: AND = conjunction, OR = disjunction, giving CNF it's name)
  - Can view as bipartite graph: variables vs clauses, with positive/negative edges, see Figure:



- There's a known polynomial time transformation of any SAT to CNF SAT
- **3SAT:** [Cook 1971]
  - clause = OR of 3 literals. i.e. clause degrees = 3
  - Most common form of CNF used.
  - **3SAT-5:** [Feige-JACM 1998; perhaps earlier?]
    - Each variable occurs in ≤ 5 clauses
    - Sometimes called max-5 occurrence 3SAT.
    - Maybe can even make each variable occur in exactly 5 clauses?
    - (Note that 3SAT-4 is also hard[1])
  - **Monotone 3SAT:** [Gold-I&C 1978]
    - Each clause is all positive or all negative.
    - Of course trivial if all clauses are all positive (negative) because can set all variables to true (false).
    - CONJECTURE (Sarah Eisenstat): Monotone 3SAT-5 is hard.

3 Beware polynomial-time variants

- CNF variants
  - **2SAT $\in$ P**
    - clause = OR of 2 literals
    - $x \vee y \qquad \equiv \neg x \Longrightarrow y \qquad (\equiv \neg y \Longrightarrow x)$
    - Solution is to check x_i, and follow all implication chains to check OK
    - **MAX 2SAT is NP-complete** [Garey, Johnson, Stockmeyer 1976]
      - set variables to satisfy k of clauses, or to maximize number of true clauses
  - **Horn SAT $\in$ P** [Horn 1951]
    - Specification of CNF SAT
    - Each clause has ≤ 1 positive literal
    - e.g. $\neg x \vee \neg y \vee \neg z \vee w$
      $\equiv \neg(x \wedge y \wedge z) \vee w$
      $\equiv (x \wedge y \wedge z) \Longrightarrow w$
  - **Dual-Horn SAT $\in$ P** [Schaefer 1978]
    - ≤ 1 negative literal in each clause
    - Easy to see in P, as one can just negate Horn SAT to get dual Horn SAT
- **DNF (Disjunctive Normal Form) SAT $\in$ P**
  - formula = OR of clauses
  - clause = AND of literals
  - $\Longrightarrow$ satisfiable $\iff \geq 1$ clause is true.
  - Thus basically always true.
- **Renamable Horn**
  - $\exists$ negation of $Y \subseteq X$ s.t. Horn

We've covered many variations of SAT, which naturally leads to the question: do you really have to remember all? Unfortunately, yes, but there's a shortcut we'll get to in another page.

[1] http://cedric.cnam.fr/~bentzc/INITREC/Files/CB11.pdf

4 Alternative clauses for 3SAT

A lot of the time, problems don't really map onto ANDs and ORs. Oftentimes have bits, but they may not correspond to logical notions of AND and OR, and the operations may not correspond either. Next two versions of SAT are in that spirit.

- **1-in-3SAT = exactly-1 3SAT is NP-complete** [Schaefer 1978]
  - clause = exactly 1 of 3 literals (x_i, x_j, x_k) is true (i.e. TFF, FTF, FFT).
  - **Monotone 1-in-3SAT is NP-complete** [omitted by Schaefer]
    - literals = variables (don't need negations).
    - Aside: the terminology is a little confusing as "monotone" here means all positive, whereas earlier monotone meant all positive *or* all negative
  - **Monotone not-exactly-1 3SAT $\in$ P** [Schaefer 1978]
    - clause = 0, 2, or 3 of 3 variables true.
    - Trivial as written because can just set all variables to false.
    - Even if set $x_1 = TRUE, x_2 = FALSE$, $x_i \Longrightarrow (x_j \wedge x_k)$, which is the same as $\neg x_i \wedge x_j \wedge x_k$, $\rightarrow$ Dual Horn
    - Note that we do not allow negations here.
- **NAE (Not All Equal) 3SAT is NP-complete** [Schaefer 1978]
  - clause = 3 literals not all the same value
  - (This forbids FFF & TTT $\Longrightarrow$ 1 or 2 true, 2 or 1 false, whereas 3SAT forbids just FFF)
  - Can think of as exactly 1 or 2 in 3SAT
  - Nice symmetry between TRUE & FALSE
  - **Monotone NAE 3SAT is NP-complete** [omitted by Schaefer]
    - no negations - variables = literals, so all literals positive

Important problems are 3SAT, 1-in-3SAT, and NAE 3SAT for lower bound proofs. When trying to get clause gadgets, we want them to be constrained somehow, and often it ends up being one of these three constraints, possibly with negations to make them happy. We don't want to fall into one of the other states that's polynomial.

Student question: what about three colors / ternary truth? In general, you'd have to go through the work to check each problem. Think they're pretty uncommon. Most common practical answer, for instance with 4 values, is to call 2 true and 2 false and hope that it works out, but you have to be careful.

5 Schaefer's Dichotomy Theorem [Schaefer-STOCK 1978]

With the right setup, all versions of SAT are either NP-complete or in P.

- formula = AND of clauses
- general clause = relation on variables.
  - Assume in CNF (unique if minimal using Karnaugh maps)
  - $\Longrightarrow$ AND of subclauses
- Claim: SAT is polynomial if at least one of 4 cases happen:
  1. Setting all variables true or all variables false satifies all clauses
  2. Subclasses are all Horn or all dual Horn

3. Relations are all 2-CNF
   - Subclause sizes ≤ 2
   - Note that this is not redundant with case 2, as we can have some Horn and some dual-Horn subclauses.
4. Every relation can be expressed as a system of linear equations over $\mathbb{Z}_2$
   - e.g. $x_i \oplus x_j \oplus x_k \oplus x_l = 0$ or 1.
   - (can just use Gaussian elimination)

- Otherwise, SAT is NP-hard (and thus NP-complete assuming checkable).

In general, there isn't really a geometric intuition, though such is obvious for case 4.

It's unclear if there's a theorem on how hard it is to recognize whether something fits into one of the four cases, though an algorithm for such would be very nice.

That's all the versions of SAT you need to know, given that we now have a universality theorem. It's not generally necessary to figure out in advance what version of SAT we're going to use, but rather we can just see what the gadgets we've come up match.

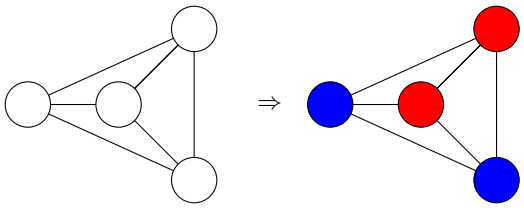
6 NP-Hardcore time

Let's do some reductions!

6.1 2-colorable perfect matching

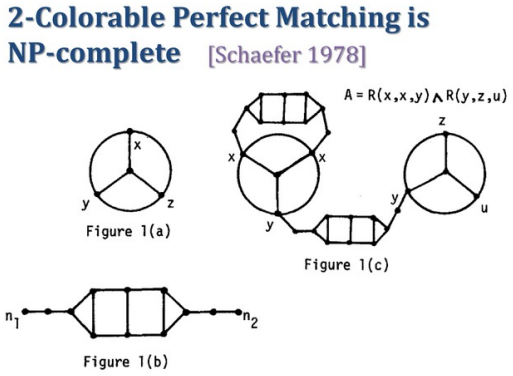
We'll prove this is NP-hard.

- Given (planar, 3-regular) graph
- 2-color vertices such that every vertex has exactly 1 neighbor of the same color



- red nodes form a perfect matching red nodes, and blue nodes form a perfect matching on the blue nodes
- can think of it as a special version of 2-in-4 SAT.

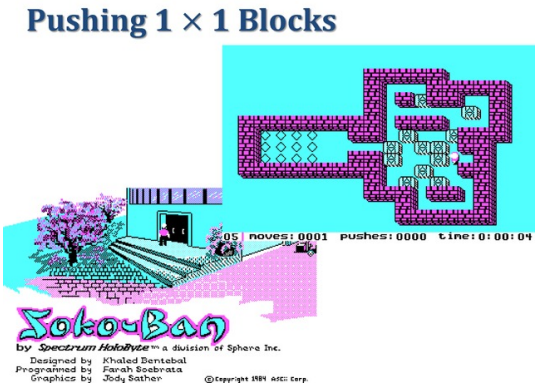
SLIDE 2 (Schaefer 1978): Reduce from NAE 3-SAT. Gadget from Figure 1(a) is K_4. Vertices X, Y, and Z, must not all be equal, so it acts like a Not-all-equal clause/gadget.



Need the ability to copy data, so that the same x_i appears in multiple clauses. Gadget from Figure 1(b) shows a clause to copy a variable. Connecting nodes with this gadget will force them to be equal. Figure 1(c) is an example of two clauses connected by a copy gadget. Simple "proof" for general graphs. Would need a cross-over gadget to make the proof work for planar graphs. To prove 3-regularity, we would need a gadget to split high degree nodes into lower degree ones, possibly with the copy gadget (suggested by Adam).

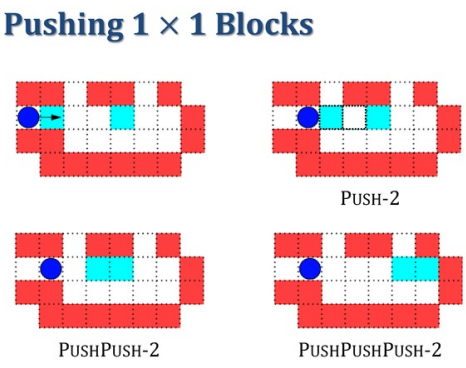
6.2 Pushing 1x1 blocks

Soko-Ban is a game (literally warehouseman in Japanese) where you, a 1x1 player, push around 1x1 boxes. Your job is to push boxes into target locations. Some boxes you can't push. Can only push 1 block at a time by one space (e.g. you'll be stuck if you push a box into a corner). (See SLIDE 3)



Lots of games have a pushing blocks component. Legend of Zelda (SLIDE 4) has a variation where the blocks are on ice so blocks will fly off to infinity unless encountering an obstacle. Goal is to cover a target space with any block.

There are other models of the game (See SLIDE 5 for figures):



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

- Normal model (upper left). Push. When you push a block, the block moves one space.
- Push-k (upper right). k is the strength of the pusher (can only push k blocks at a time).
- PushPush (down left). When you push a block, slides until you hit another block.
- PushPushPush (bottom right). When you push a block, all slide until you hit an immovable block.

There are a lot of variants, and SLIDE 6 describes known complexity in this field. Push designates how many blocks can be pushed (* designates k to be infinite). Fixed (ending F) refers to if fixed blocks are available (typically in bounded rectangle). Slide tries to capture how far things slide when pushed (PushPushPush redefines what "max" means). Not just the model of push, but the goal matters too. Sokoban has a storage goal. Some just have a path goal, to get the pusher to a place. One other variant includes

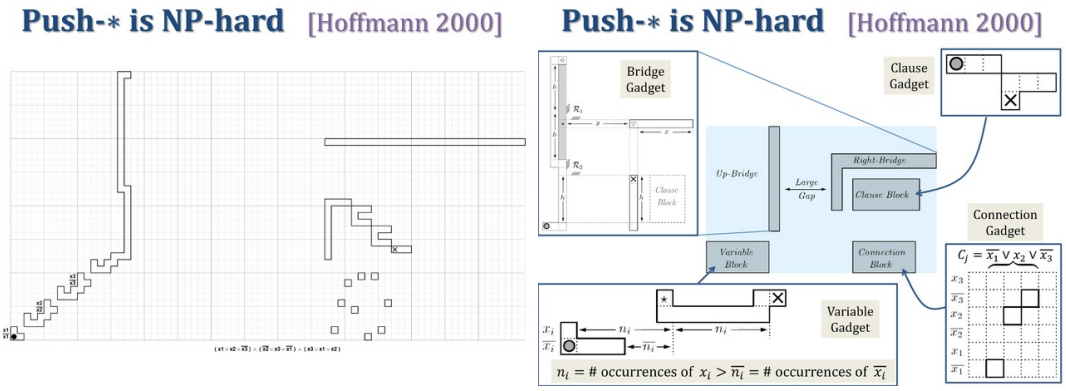
a simple path variant (ending X), where you can't cross your own path (useful in games where blocks you walked on previously disappear).

All of these variants are NP-hard, but some are known to be PSPACE-complete, and with simple paths, we can prove they are NP-complete because possible solutions must have polynomial length, so the problem is in NP.

Giving blocks variable weight seems like it would only make things harder.

6.3 Push-* is NP-hard (Hoffmann 2000)

The model is Push-* in a fixed box and we reduce from 3SAT. Refer to SLIDE 7-8. Encode SAT gadgets into the blocks. Specifically, we have a Bridge Gadget, Variable Gadget, Clause Gadget, and Connection Gadget.



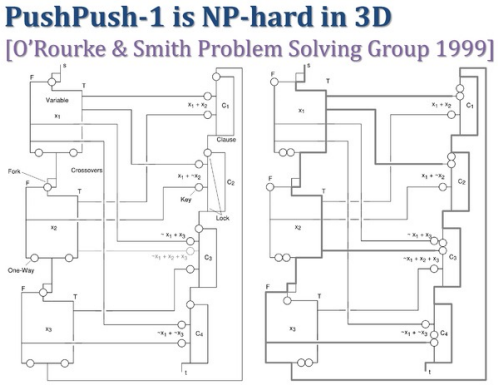
We construct a very constrained path where most of the space is occupied by movable blocks. First, we walk through the Variable Block where the pusher has an option to push rows to the right (either a positive or negative instance) into free space in the Connection Block. In general these gadgets are super tight, so there's nothing you can do except push a row to fill everything to your right; pushing both positive and negative instances of a variable only makes things worse further down the line (in the Clause Block).

The Connection Block (bottom right) is a bipartite graph encoded into a matrix with open spaces when variables (rows) connect to clauses (columns). Pushing a satisfying sequence of variables to the right will leave enough space in the Connection Block to maneuver the Clause Block later.

The Up-Bridge and Right-Bridge force the transition from variables to clauses to be one-way and to fill in the space to the left and top of the Clause Block.

The Clause Gadget allows you to make progress only when there is space below you on any of the variables in your clauses (this is why pushing both instance of a variable only hurt you, because it may block off needed empty squares).

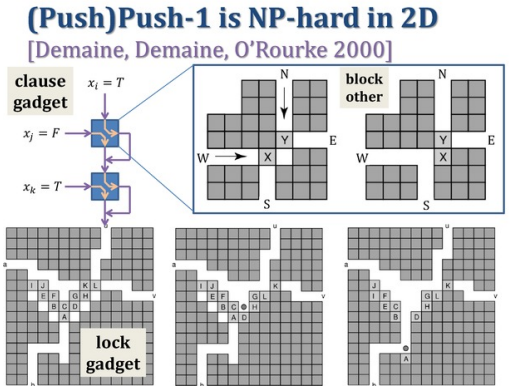
6.4 PushPush-1 in 3D

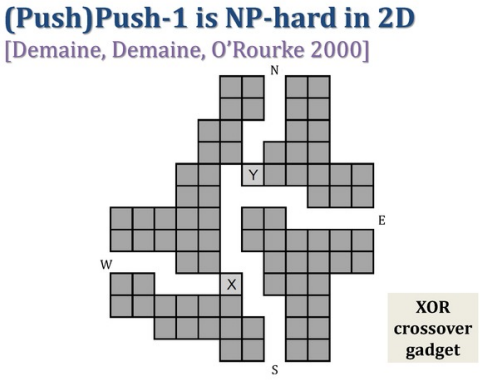


This problem involves pushing blocks with sliding in 3D (SLIDE 9). The paths are little width 1 tunnels. At each variable block, you can push a block in the True or False direction which opens one way, and closes the other way, never to be traversed again. The two blocks at the bottom of each variable gadget keep you from backtracking. At the end of the last variable, you run through all the clauses. If any of the literals that satisfy a clause were visible, you could have pushed it to the right, blocking off the left vertical tunnel in the clause gadget, allowing the top clause gadget block to be pushed down while keeping the bottom tunnel of the clause gadget open for traversal. This reduction is the basis for many of the proofs of this type; a very straight forward 3SAT proof.

Because we're in 3D, we don't need a cross-over gadget as paths can be routed without crossing.

6.4.1 (Push)Push-1 is NP-hard in 2D





For 2D, we need a lock gadget and a crossover gadget (SLIDE 10-12). For the lock gadget (bottom left), the goal is to go from A to V. If you try to go from A to V directly, it's impossible as constructed. But if you visit from U first, you can unlock the lock. We can use this lock to create a clause gadget with three possible entires. If you come down the top path, it prevents you from using the other half of the gadget and escaping along that path. As you come down, you force the gadgets to be in a particular state to force you to go back the way you came. Then, once all clauses are unlocked, you will be able to traverse through all the lock gadgets to complete the path.

Then we have the issue of a crossover gadget in 2D (See right Figure above). Going from N to S, we can push Y down, but then can't go W to E. Can only do one or the other. This is called an XOR crossover gadget, usable only once. Not what we want, but we can chain together locks with the XOR crossover to create a unidirectional cross-over gadget as shown below. You can do one of three things: N-S, W-E, or N-S-W-E. Included also are no-reverse gadgets that do not allow you to return the way you came. Then we connect variables and clauses together (same outline as in Super Mario Bros from Lecture 1). We know the order in which the crossovers happen, so we can orient the crossovers in the right orientation. Gadgets are different from Super Mario Bros, but proof structure is the same.