6.890: Algorithmic Lower Bounds: Fun With Hardness Proofs Fall 2014

Lecture 16 Scribe Notes

Prof. Erik Demaine

1 Overview

This class will come back to the games topic. We will see the results of the "Gaming is a hard job, but someone has to do it!" paper by Giovanni Viglietta[2]. It shows some metatheorems that will be used as general techniques for proving hardness. We will cover 2 main metatheorems (and some different versions of them): one for NP-hardness and one for PSPACE-hardness. They can be applied to a lot of games. We will use the term metatheorem in somewhat vague sense and not a formal theorem, because it's hard to state all the assumptions for all games. It will give a general set up for the proof.

2 Definitions

Viglietta defines an "avatar" in his proofs, but for our case, we will use the term "player." The player is the the character in the game that we can control and move around to complete objectives. One basic assumption we make about the player is that we can choose, at any time, to change the player's direction of movement.

3 Metatheorem 1

This metatheorem enunciates that if a game with a player traversing a 2D environment with a start location and:

- Location traversal (with or without a starting location or an exit location)
- Single-use paths (each path can only be traversed once)

then the game is NP-hard. Location traversal means that the player has to visit some locations in the board in order to win the level. The planarity of the problem will allow us not to worry about crossovers.

The claim of this metatheorem is that any game with such characteristic can be reduced from Planar Max-Degree 3 Hamiltonian Path. The reduction should turn each vertex in the graph into a location that must be traversed and each edge should become a single-use path. Since each vertex has degree 3 and each of the edges are single-use, once we traverse two edges to visit the vertex and leave, those edges will disappear, leaving just one edge. Now, the vertex is unreachable because if we use the third edge, the player will be trapped in that location, as there is no fourth edge out of the vertex.

Thus, we conclude that there will be a way to clear the game if and only if there is a Hamiltonian path.

We've seen reductions like this before, but with a time limit; this is another way to get the same kind a proofs.

Using this metatheorem, we can prove NP-hardness for many games.

3.1 Boulderdash

In this game, the player has a starting position and can walk around without being affected by gravity and dig in adjacent cells if they are made of earth. There are boulders are influenced by gravity and if they fall on the avatar, the player dies. The goal is to collect all the diamonds and get to the end location. You only need two gadgets: location traversal (shown in Figure 1 left) and single-use path (shown in Figure 1 center).

Figure 1 right demonstrates the single-use path gadget after it has been traversed from left to right: we push the first boulder into the pit in order to clear the obstacle; we then push the lower of the two stacked boulders over to the other pit, then push the last boulder into the final pit as part of our traversal. During this time, the higher of the two stack boulders falls down and blocks our path, since there is no pit to push it into.

(One small note is that the boulder can "rest" on the player without killing him, so pushing the lower boulder to the right and causing the player to stand under the higher boulder is permissible.)

The conclusion is that Bouderdash is NP-hard.

Figure 1: (Left) location traversal, (center) single-use path gadgets for Boulder Dash. (Right) Single-use gadget after traversing from left to right.

3.2 Lode Runner

In this game the player have to collect all the coins and avoid enemies. You can dig a hole on the ground and the monsters can fall in this hole. Eventually, the hole will refill (after some specific time) releasing the monster. The avatar cannot jump and this property is exploited in the Single-use path gadget. Figure 2 shows the gadgets required for metatheorem 1. The conclusion is that Lode Runner is NP-hard.

3.3 Zelda II

In this game you are a character called Link and this is a platform game. The location traversal is done by placing keys on certain locations. The keys are used to open doors. At the end of the level there will be exactly the same number of doors as the number of keys in the level, so in order to clear the level, we must collect all keys. The single-use paths are bridges that disappear when

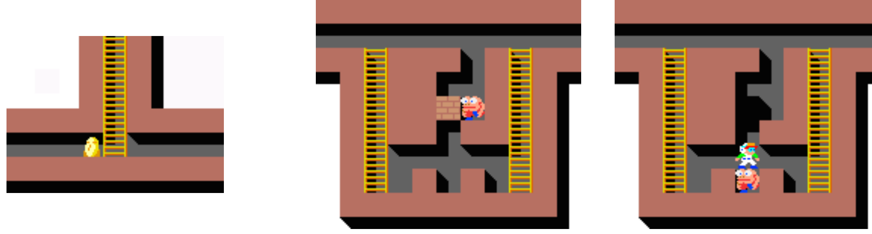

Figure 2: (Left) Location traversal gadget. (Center) Single-use path gadget and (right) it being used.

Link walks over them. An animation illustrating these two concepts can be found in the slides for the class in the course website.

The conclusion is that Zelda II is NP-hard.

4 Metatheorem 2

This is a slight variation of Metatheorem 1. It starts from the same kind of set up (player with a starting location), but requires tokens in order to traverse a path instead of single-use paths. The player collects tokens that appear in determined locations in the level, and uses the tokens to pay a toll in order to traverse some path. The idea is to simulate a single-use path with this kind of mechanism:

- We place a token at each vertex, and
- We transform every edge into a toll road that requires one token.

Thus, to get from one vertex to the next, the player must pay the token that was just acquired; if a vertex is visited more than once, there will be no token at that vertex, and the player is unable to cross over any edge.

The reduction from Hamiltonicity follows in the same way as in Metatheorem 1.

4.1 Pac-Man

We can use Metatheorem 2 to prove Pac-Man NP-hard. In order to win in Pac-Man, you have to collect all the dots, which will also function as our tokens. Figure 3 shows the degree-3 vertex with the token (dot) and the toll road (path with ghosts). Eating a token causes the ghosts to change state, so that Pac-Man can eat the ghosts; after some time, though, the ghosts revert to becoming harmful to Pac-Man. Also, whenever a ghost changes its state, it will change it direction of movement.

The ghosts that are “eaten” revive after a while and appear inside a “cage,” which they then leave and continue moving around in their original paths.

Thus, the only way to traverse an edge is to consume the token and the ghost along the chosen exit edge. For the sake of argument, we will assume that the ghosts change state long enough for

Pac-Man to exist safely, but short enough so that if Pac-Man even comes back to the same area of the map, the ghosts will have reverted states or respawned. Due to this, each edge can be traversed if and only if Pac-Man consumes the token; once the token is consumed, Pac-Man cannot return to this "vertex" again, since he won't be able to pay the toll anymore.

Therefore, Pac-Man is also NP-hard.

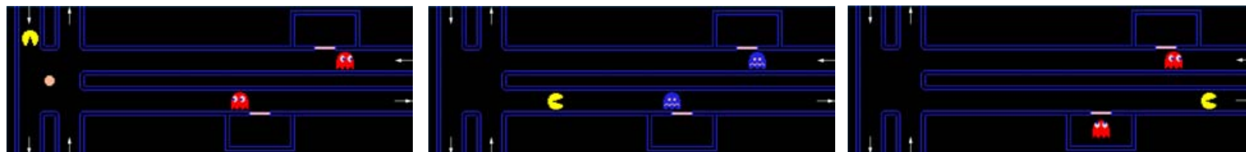

Figure 3: Token+Toll road gadget for Pac-Man.

5 PSPACE

PSPACE is the set of problems solvable in polynomial space. This set is contained in EXP because there are at most exponentially many states in a polynomial space machine and is contained in NP because you can try all possible options for the guess at each possible answer (not keeping this information) and keep only the state in polynomial space.

Also, PSPACE equals NPSPACE (non-deterministic polynomial space) because every NPSPACE machine with a PSPACE machine at most blowing up the space by a square factor. This is useful to show that a problem is in PSPACE because we have the freedom to use non-deterministic algorithms (guessing whenever you like).

5.1 PSPACE-complete problems

In order to show that a problem is PSPACE-hard, we need a set of problems known to be PSPACE-complete (to reduce from). The classical problem is to simulate a polynomial space algorithm (or simulate a linear space Turing Machine). This problem not very useful for hardness proofs.

A simpler problem to reduce from is QSAT (also known as QBF or TQBF), which stands for Quantified Boolean Satisfiability (the Q means "quantified"). The statement of the problem is:

Given a fully quantified boolean formula, is the statement true?

An example of a quantified boolean formula is

$$\forall x, \exists y : (\overline{x} \vee y) \wedge (x \vee y)$$

The above formula is an example of Q2SAT that is true and the solution is to make $y = x$. Q2SAT is NOT PSPACE-complete, but Q3SAT is. The difference from traditional 3SAT is that in addition to the familiar "$\exists$" condition, we also have several "$\forall$" conditions to satisfy as well.

For our purposes, we can assume that the quantifiers will be always in the front of the statement (prenex form) and there will be an alternation of $\forall/\exists$.

5.2 Schaefer-style dichotomy theorem

This theorem states which forms of QSAT are PSPACE-hard. This is a Schaefer-style dichotomy theorem.

- QSAT $\in$ P $\iff$ Horn, Dual Horn, 2-CNF or X(N)OR
- QSAT is PSPACE-complete otherwise.

As with regular k-SAT, the planar versions of QSAT are also hard: we can start with the same crossover gadget as seen in Lecture 7 that forces some variables to be the same, and then also have it create new variables that must also be quantified. We can do this by adding a $\exists x_i$ for every newly created variable x_i.

Notably, Planar 1-in-3QSAT is still hard, and Planar NAE 3QSAT is still easy.

6 Metatheorem 3

The third metatheorem presented stated that a game where you have a player who needs to traverse a planar graph from start to finish, and has door and pressure plate objects, is also PSPACE-complete. A door can be considered an edge that exist only if a certain condition is met. There are two types of pressure plates – an "open" pressure plate which satisfies the condition of the door, and a "close" pressure plate which causes the door condition to not be satisfied.

(Keep in mind that these pressure plates simply cause the door to open or close, but do not require constant pressure to keep them in that state, i.e. unlike Portal's door mechanics.)

6.1 Reduction from Q3SAT

The lecture slides 9 and 10 provide the visuals for this reduction. The idea is as follows:

- We start at the position labeled "start," and continue along the path.
- Wherever there is an $\exists$ quantifier, we fix a value for that variable.
- Wherever there is a $\forall$ quantifier, we will check both possible values by:
  1. Traversing towards the clauses (top branch in the slide 9 diagram), we remember that we have seen this variable once, and set its value to true.
  2. Traversing back from the clauses (bottom branch in the diagram), we check to see if this variable is true: if true, set to false and loop again; if false, continue along the branch.

Remember that we will continue along the bottom only if the quantifier is satisfied; e.g. if one value fails to satisfy the formula for a $\forall$ quantifier, the second loop is no longer necessary, as we already know that the $\forall$ quantifier cannot be satisfied.

Now, we let a door's state determine the value of its *literal*: an open door indicates that its represented literal is selected. Thus, for a variable x, we need two doors for it: x and $\neg x$. Note

that, when we say an x door, we actually mean all doors labeled by x such that an x-open switch opens all the x doors, and so on for closing and $\neg x$.

Then, a clause is a hall with the doors possible to the next hall, one for each literal; if any of the three doors is open, the clause is considered to be satisfied.

Finally, we need to come up with our quantifiers:

- Existential quantifier: two branching paths, that mutually close each other, and turn the variable on or off
- Universal quantifier: snake-like path, with middle gate that "counts" how many times we've gone through

The clause gadget and the quantifier gadgets are both diagrammed in slide 10. One key point to note about the quantifier gadgets is that they prevent deadlock by requiring the path to open one door contain pressure plates to close the other doors such that the progression must move forward.

Note that, by construction, a solution will take exponential time: at least $O(2^U)$ time for U universal quantifiers. However, the reduction process should still take polynomial time.

6.2 FPS games

This metatheorem allows many FPS games such as Quake to be easily proved PSPACE-complete, by simply designing the maps and puzzles using pressure plates and doors as described in the reduction above.

6.3 RPG games

Additionally, this metatheorem allows one to prove many RPG games (such as Eye of the Beholder) to be PSPACE-complete, using the same idea of designing the maps and puzzles correctly.

6.4 SCUMM engine

Many adventure games can also be proven PSPACE-complete in this way. One rather large categorization of such games – the SCUMM engine – can be shown to be PSPACE-complete allowing all games that use it to be shown to be as well. Some SCUMM engine-based games are Secret of Monkey Island, Maniac Mansion, Space Quest IV.

6.5 Prince of Persia

In this game, the player is given the ability to jump. So, in order to ensure that pressure plates are always pressured when the player passes over that tile, we put the pressure plate on top of a very high wall, such that the player can only jump that high, so the player must touch the pressure plate (which is important to ensure that the player must make progress in the game).

Additionally, the broken video on slide 14 of the lecture notes is supposed to show a player jumping down a ledge, literally pushing a pressure plate into the wall on the right, and then jumping left across a gap to get to the other side.

7 Metatheorem 4

Metatheorem 4 expands upon metatheorem 3 to allow the use of buttons instead of pressure plates. These buttons do not need to be pressed and open/close 3 doors at once (and in fact it was recently shown that 2 doors at once will work, but this has not yet been published).

The reduction for 3 doors at once involves treating a button as 3 pressure plates put together.

7.1 Examples

Some examples of games that fall under this pattern are Sonic the Hedgehog, The Lost Vikings, and Tomb Raider.

8 Metatheorem 5

Metatheorem 5 is a further generalization that relies on doors and crossovers to show PSPACE-hardness.

To do this, we show that we can use doors and crossovers to create an environment that provides the following three types of paths: a traverse path (whereby the player can only pass if the door is opened), an open path (which allows the player to open the door) and a close path (which forces the player to close the door).

This result can be found in the 'Nintendo paper' [1].

8.1 Legend of Zelda: A Link to the Past

In Legend of Zelda, we create a traverse path with just a labeled door: if the labeled door is open, then we can traverse; if not, then we can't.

One interesting caveat with Legend of Zelda is that the game has only toggles: it opens all the connected doors if they're closed, and closes them if they were open. This means that the open and close paths are somewhat trickier: the general idea is for the open and close paths to lead to directed teleporters to the appropriate halls (seen at the bottom of slide 19). Then, to toggle the door, we have to traverse from the direct that we want to toggle to, and then hit the toggle in a secluded room, for which the only way out is through the directed teleporter right next to it, which takes us back to a hall.

Since we want to initialize all doors to the closed state, we need to create an initializer for this purpose; basically, we just create a chain of gadgets that will toggle all the doors to be closed. At the very end, we have a crystal that can be broken, which will active the inactive toggles, and

deactive the active toggles. There will be only one crystal, and it serves the purposes of destroying our initializing paths and enabling a one-way gadget. To make sure that it doesn't appear at the wrong place, we ensure that it appears only between the initializer and final traversal gadgets.

8.2 Donkey Kong Country 1, 2 and 3

Donkey Kong Country another Nintendo series of games that can be shown to be PSPACE-complete using metatheorem 5. Unfortunately, DKC 1, 2, and 3 have mutually exclusive "features," so we will have to address each one individually.

8.2.1 Donkey Kong Country 1

In the Donkey Kong Country games, there are bees. If you touch a bee, you die, so the goal is to not die. In DKC 1, there is a tire object: if you land on the tire, you bounce back up. For a traversal, we use a barrel shot straight down, so landing on a tire would cause us to be stuck in the game (and thus not win).

Furthermore, we will introduce stationary bees as obstacles, and moving bees (highlighted in red on slide 21 – not to be confused with actual red bees in the game), which move in a deterministic pattern, demonstrated by the arrows.

To open the door, we come in from the open path, jump across the ledge, and push the tire just up the hill so that it doesn't roll back down. Note that the giant box of bees that are labeled "open" move in a left-down-right-up pattern, so that after pushing the tire up the hill, we can still run back down through the traversal path.

To close the door, we come in from the close path, and push the tire down the hill with a slight nudge, and scramble up a rope to exit. One caveat here is that, in the real game, we'd either have to make the bees slower, or make the climbing faster.

8.2.2 Donkey Kong Country 2

In DKC 2, instead of tires, we have balloons and air currents (the gray steam near the bottom of the map). The balloons float on top of the air currents, and in order to traverse, we have to move it out of the way of the traversal path.

To do this, from the open end, we can jump on the balloon and move it away from the air currents in the middle so that it drops down to point A. We then escape through the entrance of the open path.

To close, we enter from the close path, drop onto the balloon, and move it back into the current before using the barrels to propel ourselves out of the region.

8.2.3 Donkey Kong Country 3

In DKC 3, instead of tires and balloons, we have tracking barrels, which the player can slide around sideways once the player land in one, and it always shoots up. The caveat is that the barrel will

follow the player if the player tries to jump out of it, so the tracking barrel effective prevents the player from going down.

Thus, the open state is for the barrel in the big gap of slide 23 to be on the left: if the player enters to traverse, the player can drop into the barrel, get shot up, and exit through the traversal path.

To close, we enter from the close path, jump into the barrel, and slide it all the way to the right. Since the barrel tracks us wherever we go after we land in it, we are guaranteed that, if we leave through the close path, the barrel must be on the right.

To open, then we just drop into the barrel from the open end, slide to the left, and shoot ourselves out through the open path.

8.3 Super Mario Bros.

Super Mario Bros is also PSPACE-hard. Using the diagram on slide 24, we see that the traversal path is on the left, the open path is on the bottom, and the close path is the entire right side.

The intuition behind the setup is: we need something that the player can't pass through, but an obstacle can. So, we break two rules in the game itself and have (1) a length-1 firebar (a.k.a. a fireball) to separate the traversal and close paths, and a spiky to block a path.

To close, we enter from the close side, and if the spiky is on the right, then we go under the spiky, and at the right time, we knock up up and over the fireball to the other side, so that we can traverse the close path.

To open, we enter from the open side, knock the spiky over to the right, and leave again.

Then, when we traverse, we can traverse if and only if the spiky isn't on the traverse side.

8.4 Lemmings

Lemmings is an old real-time strategy game where the player is in control of a bunch of Lemmings, and can imbue any Lemming with jobs. For our purposes, we will have two jobs: a builder (to build bridges) and a basher (to cut through dirt, but not steel).

Although many results for Lemmings exist, we will show PSPACE-completeness using an exponential amount of builders, bashers, and time.

The standard Lemming is a Walker, and its mechanics are documented on slide 28: in brief, Walkers can climb up out of ditches but only if they're not too tall.

One kind of special Lemming that we will use is a Basher, who can cut through dirt but not steel. The mechanics are documented on slide 29, and a Basher will stop bashing once the steel check finds steel, or a solidarity check finds no more dirt. In the slides, the metal is represented by light-blue squares. What is interesting to note is that disparity between the steel and the solidarity checks.

The last kind of special Lemming that we will use is a Builder, who can build bridges over gaps (otherwise the Lemmings will die from falling between the gaps). The mechanics are documented on slide 30, and again, there are solidarity checks for where to lay the bridge pieces, and whether the Lemming can keep going forward and up, by using a solidarity check near the head.

In order to apply metatheorem 5, we need to show the primitive paths that we will use: a rising path, a falling path, and rise-and-double-back path, and fall-and-double-back path, and a crossing path. To prevent Lemmings from effectively running away, we place steel pretty much everywhere that isn't marked with a red crossed-box or a gray platform on slide 31.

Now, we will need a fork (a.k.a. crossover) gadget, to allow for traversals of two different paths. This is achieved by having a gap for lemmings to fall through: if they fall through, they'll go through one paths; if they build a bridge instead, they'll go through another path (which also requires a Basher to clear some obstacles, which are placed to prevent builders from going crazy and building a stairway to heaven).

Finally, our door gadget will be represented as such on slide 33: an open door will have a gap in the middle, while a closed door will have a bridge in the middle that will prevent traversal from the top to the bottom. To open the door, we bash the bridge apart using a Basher; to close the door, we layer a bridge on top of the gap using a Builder. Since we have a limited (but exponential) number of Builder and Basher upgrades available to us, we must use each one wisely.

Thus, with our door and crossover gadgets, we have shown that Lemmings is PSPACE-hard.

References

[1] Greg Aloupis, Erik D. Demaine, and Alan Guo. Classic nintendo games are (np-)hard. *CoRR*, abs/1203.1895, 2012.

[2] Giovanni Viglietta. Gaming is a hard job, but someone has to do it! *Theory of Computing Systems*, 54(4):595–621, 2014.

MIT OpenCourseWare
http://ocw.mit.edu

6.890 Algorithmic Lower Bounds: Fun with Hardness Proofs
Fall 2014

For information about citing these materials or our Terms of Use, visit: http://ocw.mit.edu/terms.