Class 2 Scribe Notes Notetakers: Jason Ku, Chelsea Voss

6.890

*Instructor: Erik Demaine* *2014-09-09*

## 1 Useful Problems for Hardness Reductions

This lecture mostly focuses on using 3-Partition to solve 2+ problems by reducing to number problems. The basic idea is to think of your numbers as integers – fixed-point or rationals are basically integers. Most proofs today will be reduction from 3-Partition. All about NP-hardness, btw. But before that, we'll talk about 2-Partition, defining these problems, then do some fun reductions. Note that throughout this lecture, Erik actually means "multiset" instead of "set" unless otherwise stated – i.e. numbers are allowed to repeat.

### 1.1 2-Partition

Given $n$ integers, $A = \{a_1, a_2, \ldots, a_n\}$, partition them into two groups: $A_1$ and $A_2$, where $A_1 \dot\cup A_2 = A$ such that $\Sigma A_1 = \Sigma A_2 = \frac{1}{2}\Sigma A = t$. Here we let $\dot\cup$ denote disjoint union[1] (hence partition).

For each integer we have a binary choice: does it go in $A_1$ or $A_2$? This question is NP-hard and was one of Karp's problems from 1972. Karp wrote a paper with a zillion[2] NP-hardness proofs from SAT to things like 2-Partition. If you're interested, read Garey and Johnson[3] for coverage of it.

Sideline: Subset Sum is a generalization of this problem which is still NP-hard. Given $n$ integers, $A = \{a_1, \ldots, a_n\}$ and a target sum $t$ want to find a subset $S \subset A$ such that $\Sigma S = t$. It is easy to think of an instance of this problem as a partition, although it's a generalization. Reducing from Subset Sum that we can reduce from 2-Partition. 2-Partition to Subset Sum is a strict generalization – not given $t$ – but we are essentially choosing a subset $A_1$ whose sum is $\Sigma A/2$. Sometimes it can be easier to think about Subset Sum.

### 1.2 3-Partition

3-Partition is Erik's favorite NP-hard problem: given integers $\{a_1, \ldots, a_n\}$ and no target sum, partition them into (not three! but) $n/3$ sets $A_1 \dot\cup A_2 \dot\cup \ldots \dot\cup A_{n/3} = A$ of equal sum: $\Sigma A_i = \Sigma A/(n/3) = t$. If this were about partitioning into 3 groups, it wouldn't be very interesting or much different than 2-Partition – but it's $n/3$ instead. We will assume $n$ is divisible by 3; otherwise the answer to the input is no.

**Fun Fact about 3-Partition**

On average the sets will be of size three, but you can make all of the sets have size three by assuming each $a_i \in (t/4, t/2)$ (open interval). Then clearly $|A_i| = 3 \forall i$ if a solution exists. It's up to you whether to ask for the $a_i$'s to have size three or not, but both formulations of the problem are hard. Using a trick, you can add "$\infty$" to each $a_i$ – any solution before is still a solution (think of $\infty$ as some arbitrarily large number like $n^{100} \cdot \max A$), but all the $a_i$'s become roughly equal to each other and arbitrarily close to $t/3$. Let's talk about a couple of related problems to 3-Partition, and why we're talking about it.

#### 1.2.1 Numerical 3-Dimensional Matching

This problem has a funny name – we will get to that second, don't worry for the moment. This problem is a closely related, specialization of 3-Partition. Given $A = \{a_1, \ldots, a_n\}, B = \{b_1, \ldots, b_n\}, C = \{c_1, \ldots, c_n\}$, partition $A \dot\cup B \dot\cup C$ into $n$ triples, $S_i \in AxBxC$ of equal sum $t = (\Sigma A + \Sigma B + \Sigma C)/n$. This is almost the

[1] http://en.wikipedia.org/wiki/Disjoint_union
[2] exaggeration, really 21 http://cgi.di.uoa.gr/~sgk/teaching/grad/handouts/karp.pdf
[3] http://en.wikipedia.org/wiki/Computers_and_Intractability

same problem, but we changed what $n$ means by a factor of three and now you have to choose one integer of each "color" ($A$ red, $B$ green, $C$ blue) to constitute a valid subset.

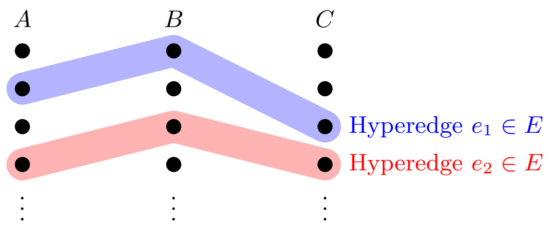
Why is this problem in some sense simpler? We can reduce from 3-Partition. We will use a useful technique for playing with these number problems when reducing from 3-Partition to Num 3-Dim Matching. We want to force this $A, B, C$ property. To convert an instance of Num 3-Dim Matching to 3-Partition add big numbers!

- add $\infty$ to each $a_i$
- add $3\infty$ to each $b_i$
- add $9\infty$ to each $c_i$

for some $\infty \approx 10 \times \max(A \cup B \cup C)$. The new $t'$ becomes $t + 13\infty$. This forces us to pick one from $A$, one from $B$, and one from $C$. We must show that the infinities can be treated algebraically despite being so large. Note that we require that the sets be of size exactly 3 as part of the 3-Partition specification for this reduction.

### 1.2.2 3-Dimensional Matching

The above are also related to another problem, called 3-Dimensional Matching (abbr. 3DM). In 3DM, we have a tripartite hypergraph with vertices, $A \dot\cup B \dot\cup C$, where $|A| = |B| = |C| = n$ and hyperedges $E = A \times B \times C$. The problem is to find $n$ disjoint edges that cover all of the vertices.



### 1.2.3 Exact Cover by 3-sets

Another related problem is Exact Cover by 3-sets (X3C) as discussed in the literature [Garay and Johnson][4]. It is a generalized version of the above: nontripartite, but any 3-uniform hypergraph with $n$ vertices – every edge has cardinality 3. The goal is to find $n/3$ disjoint hyperedges. This is a 3-Dimensional Matching problem, converted into a graph (forgetting there are numbers) – but we draw a hyperedge exactly when $a_i + b_j + c_k$ equals the sum (refer to Num 3-Dim Matching). Because tripartite hypergraphs are 3-uniform, X3C is a strict generalization of 3DM.

## 1.3 Strong vs. Weak

Back to the issue of 2-Partition vs. 3-Partition, we explore an important distinction – weak vs. strong NP-hardness. There are two types of NP-hardness for number problems when we have integers as input –

- Weakly NP-hard: NP-hard "in the intended way". Usually we think of NP-hardness in terms of how hard your problem is, in terms of the encoding size ($n$) of your input. Thus, we allow numbers to have exponential value because even when a value is exponential, the encoding of the value is polynomial (read in binary: $\lg 2[\text{ number} = 2^{n^c}] = n^c$, which is polynomial.) Polynomial number of bits is in some sense a reasonable encoding.
- Strongly NP-hard (Stronger notion): NP-hard even when problem is restricted to numbers of polynomial value (with respect to $n$, the number of numbers).

[4]Michael R. Garey and David S. Johnson. 1990. Computers and Intractability; a Guide to the Theory of Np-Completeness. W. H. Freeman & Co., New York, NY, USA.

Note that Strongly NP-hard implies Weakly NP-hard. 2-Partition and Subset Sum are Weakly NP-hard (and are not Strongly NP-hard under the assumption that $P! = NP$). On the other hand, 3-Partition, Num 3-Dim Matching, 3DM, and X3C are all Strongly NP-hard.

Generally, it is natural to assume that your inputs are reasonably encoded – binary, ternary, etc. anything bigger than 1, i.e. don't use unary. But today, we're going to use unary a lot to talk about Strongly NP-hard problems. So, note: Strong NP-hardness is "My problem is so hard that even if I encode my numbers in unary, it's still NP-hard." If you can prove Strong, you should. It's better. Of course Weak NP-hardness is still okay.

## 1.4 Pseudopolynomial, Weakly Polynomial, Strongly Polynomial

There are some corresponding notions of strength on the algorithms side.

- Pseudopolynomial – omit the log in the representation so that the algorithm is polynomial in $n$ and the largest number in unary.
- Weakly Polynomial – This is "the usual notion of polynomial" so that the algorithm is polynomial in $n$, and log( largest number ).
- Strongly polynomial – the algorithm is polynomial in $n$ the number of numbers (usually relevant in models of computation where e.g. arithmetic is free)

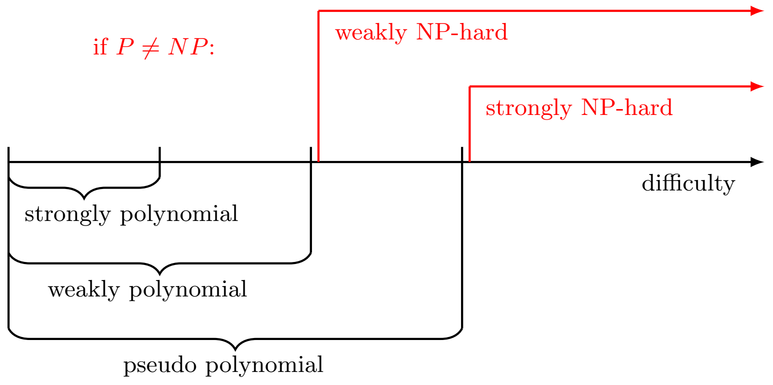
What we care about most is the distinction between Pseudopolynomial and Weakly Polynomial:

- Pseudopolynomial: polynomial when numbers are written in unary
- Wealky polynomial: polynomial when numbers are written in binary

Comapared to:

- Weakly NP-hard: no Weakly Polynomial time algorithm (but there might be a Pseudopolynomial time) algorithm
- Strongly NP-hard: no Pseudopolynomial time algorithm (and therefore no Weakly Polynomial time) algorithm

Here is Erik's favorite diagram applied to these concepts. Draw a difficulty axis: but now instead of $P$, we have Weakly Polynomial, Strongly Polynomial, Pseudopolynomial.



In general, strongly NP-hard is a better result, because it is more restrictive.

- 2-Partition is weakly NP-hard and Pseudopolynomial (therefore not strongly NP-hard)
- 3-Partition is strongly NP-hard (therefore not even Pseudopolynomial)

## 2 Reductions!

Let's do some reductions, shall we? It's NP-hardcore time! Just to warn you, this first one is going to be a bit trivial

### 2.1 Multiprocessor Scheduling

Given $n$ jobs, with completion times $a_1, \ldots, a_n$, and $p$ processors, each processor sequential and identical with each job running to completion on a single processor, the decision version goal is to finish all jobs in time $\leq t$. The claim: this problem is NP-hard of some variety.

#### 2.1.1 Reduction from Partition (aka 2-Partition)

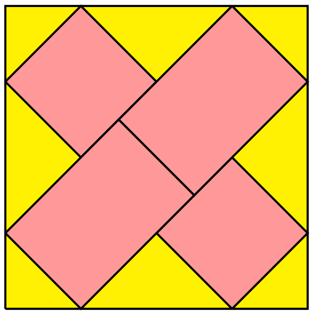
Imagine $p = 2, t = \Sigma A/2$. Problem instance looks like a 2-Partition instance so the problem is at least Weakly NP-hard.

#### 2.1.2 Reduction: from 3-Partition

Imagine $p = n/3, t = \Sigma A/p$. Problem instance looks like a 3-Partition instance so the problem is at least Strongly NP-hard (comes from Garay and Johnson paper introducing 3-Partition).

### 2.2 Rectangle Packing

Given $n$ rectangles and target rectangle $B$, do they fit into $B$? Rotation and translation allowed but rectangles must be disjoint from each other (cannot overlap) so that they fit in $B$.

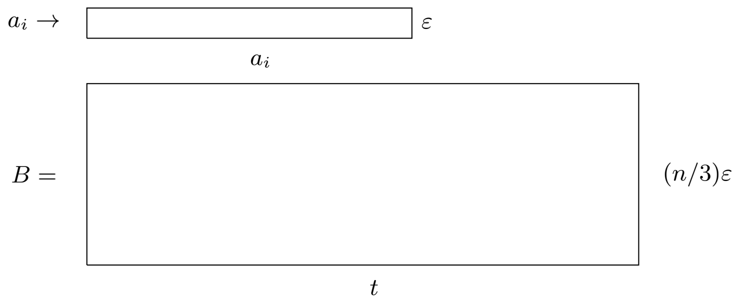


This problem is strongly NP-hard, but it is not known if this problem is even in NP (OPEN) because it is complicated to encode rotations efficiently. Let's look at a special case: packings should be exact – no gaps. Rotation must be integer $\times 90°$, so constant number of them. Translations are integral also given exact packing , so there are succinct encodings implying that this problem is $\in$NP. We no prove that both of these problems are strongly NP-hard.

#### 2.2.1 Reduction from 3-Partition to Rectangle Packing

Take each $a_i$ and make it into an $a_i \times \varepsilon$ rectangle. Let $B$ be an $(n/3)\varepsilon \times t$ rectangle (note: $t = \Sigma A/(n/3)$). To fix the rotation issue, we multiply by a $\varepsilon$ small number, so that $(n/3)\varepsilon \ll 1 \Longrightarrow \varepsilon \ll 3/n$. Then we scale up everything by $n/3$ so that everything is an integer. See Slide 2 for an example. Additionally, we add a constant $n/3$ to each $a_i$ to make the width $t + n$ instead. Now we know that rotation is not going to be possible. From [Demaine and Demaine, 2007][5].

[5]http://erikdemaine.org/papers/Jigsaw_GC/paper.pdf



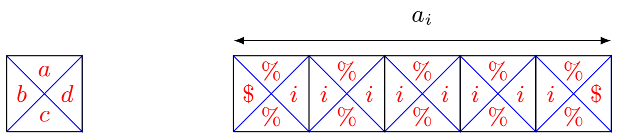
Now time for some puzzles!

## 2.3 Edge-Matching Puzzle

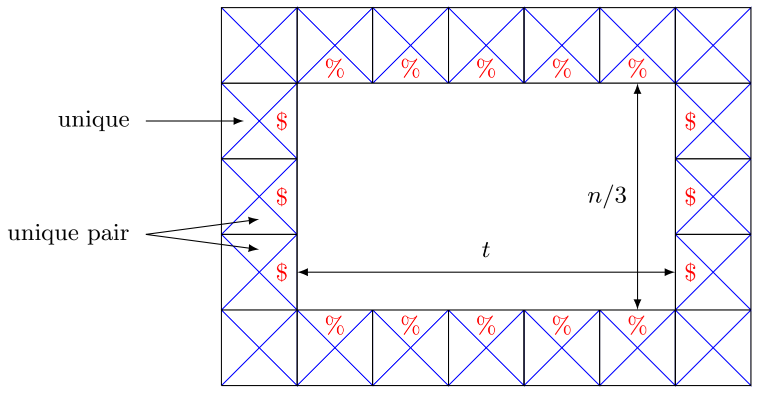
This puzzle goes back to the 1890s (See slide 3). The goal is to pack triangles with half frogs on them into a larger triangle such that the cranial and caudal ends of each frog match up (front, hind). An alternative Edge-matching puzzle are Wang tiles that are square. The model is there are $n$ unit squares (tiles), each one with 4 colors $(a, b, c, d)$, with one color per edge; and also a target rectangle, and we want to pack the squares into the rectangle with colors matching. Eternity II, is also an edge matching puzzle, currently still unsolved, with US \$2,000,000 prize in prize money ("in addition to the \$2,000,000 you would make if you also proved $P = NP$"). There are lots of almost-matches out there: e.g. Louis Verhaard's 467 out of 480, and a posted "Solution" (which involves tile duplication). Erik: "so there's your motivation for solving edge matching." The trick is we can't solve the problem generally in polynomial time unless $P = NP$.

### 2.3.1 Reduction from 3-Partition to Edge-Matching Puzzle

We reduce from 3-Partition to Edge-Matching Puzzle using a Rectangle Packing-like strategy. How can we convert $a_i$s into something for edge matching. $a_i$ becomes a long strip ($n/3 + a_i$ units long) where touching colors on the inside are color $i$ – so as to force this rectangle to be built (since there are only two "end pieces" for color $i$). To prevent rotation, colors on top and bottom are different from colors on sides (colors % and \$ in this instance).



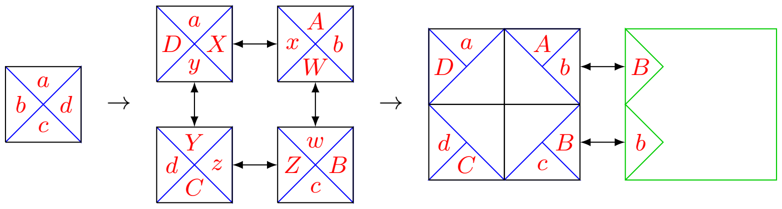
Then, build a "frame" – hide the boundary, so that no tiles could match the boundary by forcing a bunch of unique colors to all be at the edge. Frame is $n/3$ tall and $t$ wide.



Are there any numbers in this problem, as inputs? The colors need to be represented as numbers (although they are only compared, never added). Total number of different colors $\leq 4n$, anyways. So it wouldn't make sense to say this is Strongly NP-hard because the problem can really only be thought of in a unary representation. If we trying to do the same reduction with 2-PARTITION, everything would break. The $a_i$s have to be exponential in value meaning we get exponentially many tiles and exponentially many different colors in the boundary ... all bad. We need a polynomial size output, but we get an exponential sized edge puzzle – so that would not be a valid reduction. In 3-PARTITION, we are representing the numbers in unary so everything remains polynomial in size during the reduction.

## 2.4 Signed Edge-matching

Next puzzle: Signed edge-matching (Like the frog puzzle). Here, lowercase and uppercase letters for colors, and e.g. b matches with B. We can reduce from "Unsigned" to Signed Edge-Matching using this gadget:



Under this transformation, we now have new "colors" from pairs of upper and lowercase letters so that sign does not matter. Interior colors are unique pairs to force their assembly. Through this reduction, we see that the hardness is the same for Signed and Unsigned Edge-Matching puzzles.

## 2.5 Jigsaw Puzzles

For Jigsaw Puzzles, each edge is either straight, pocket, or tab. Pockets and tabs can be slightly different in shape. We can allow for some non-matching pockets and tabs to have same shapes (ambiguous mates). To make it NP-hardcore, we'll assume there are no images on the pieces (note that these types of blank puzzles are sold, but typically without ambiguous mates). But that's just a Signed Edge-Matching puzzle! Transform lowercase letter into pockets and uppercase letters into tabs, with unique colors patterning the straight boundary edges (See Slide 10). We've reduced 3-PARTITION to Edge-Matching, Edge-Matching to Signed Edge-Matching, and Signed Edge-Matching to Jigsaw puzzles. Jigsaw puzzles will now be reduced to... polyomino packing!

## 2.6 Polyform Packing Puzzles

Polyform Packing are the medium for the Eternity I Puzzle (See Slide 11). These puzzles are alse NP-hard via reduction from Jigsaw puzzles (Demaine and Demaine 2007)[6]. To reduce, take a jigsaw piece, turn it into roughly a square. Encode colors using binary and represent $4n$ different colors with binary bumps on the edge of the squares. We use $\log 4n$ bits along the edge by indent or outdenting accordingly (See Slide 13). They will fit together and make a nice clean seam if the colors match. The construction is blown up by a logarithmic factor $\log 4n$, but the number of pieces stays the same. Now, we reduce Polyomino Packing to Edge-Matching ... by making a polyomino out of those edge pieces (See Slide 14).

[6]http://erikdemaine.org/papers/Jigsaw_GC/paper.pdf

$$\begin{tikzcd}
\text{3-Partition} \arrow[dr] & & \\
& \text{Edge Matching} \arrow[dr] & \\
\text{Polyomino Packing} \arrow[ur] & & \text{Signed Edge-Matching} \arrow[dl] \\
& \text{Jigsaw} \arrow[ul] &
\end{tikzcd}$$

Fun open problem: That reduction used $\log^2 n$ pieces. Can you use $\log(n)$ area instead? One last hardness proof.

## 2.7 Square Packing

In a similar vein: What about packing squares into a square? Given a bunch of squares and a target square, the decision problem is can they fit? Squares can only rotate by a multiple of 90° (aka they can't rotate). Can we reduce from 3-Partition to Square Packing? This is a result from [Leung, Tam, Wong, Young, Chin 1990]. Take each $a_i$ number, add this huge number $B$, make the height $3B+t$ (where $t$ is the target for the $a_i$s' sum). So the little bit of extra has to add up to $t$. Width of the big rectangle is $(B+t)(n/3)$. The maximum amount of unused space (slop) is at most $t(3B+t)(n/3)$, i.e. (width)(height)(number of these things). This is less than $B^2$ if $B > 2tn$. In order to pack these squares into a square, instead of into a rectangle, we create a rectangle gadget (See Slide 14). Make it so that the extra space is a rectangle and then scale up by $3B+t$. If you want to completely pack the target square, you can compute how much extra slop there is and add in a bunch of 1x1 tiles so that you can fill in the extra space, making it cleaner and disallowing rotation.