CMSC 858F: Algorithmic Lower Bounds
Fall 2014
Puzzles and Reductions from 3-Partition

Instructor: Mohammad T. Hajiaghayi
Scribe: Thomas Pensyl

September 9, 2014

1 Overview

In this lecture, we first examine several classes of NP-hardness and polynomial time algorithms which arise from differences in how integers are encoded in problem input. We then look at the 3-partition problem, which is very useful for proving the strongest notion of NP-hardness. Finally, we use reduction from 3-PARTITION to prove NP-hardness for a handful of problems, including a set of 4 packing type puzzles which we also show equivalent.

2 Types of NP-Hardness

Consider a number problem, that is, a problem whose input includes one or more integers. Such a problem may be either weakly or strongly NP-hard, depending on the encoding used:

- A **weakly NP-hard** problem is NP-hard when input numbers are allowed to be *exponentially* large in the input size n. This is the normal meaning of NP-hard, and corresponds to the numbers being encoded in binary, since n bits are sufficient to represent numbers as large as 2^n.
- A **strongly NP-hard** is NP-hard even when we restrict the input numbers to be only *polynomially* large in the input size n. This corresponds to requiring that any numbers be encoded in unary.

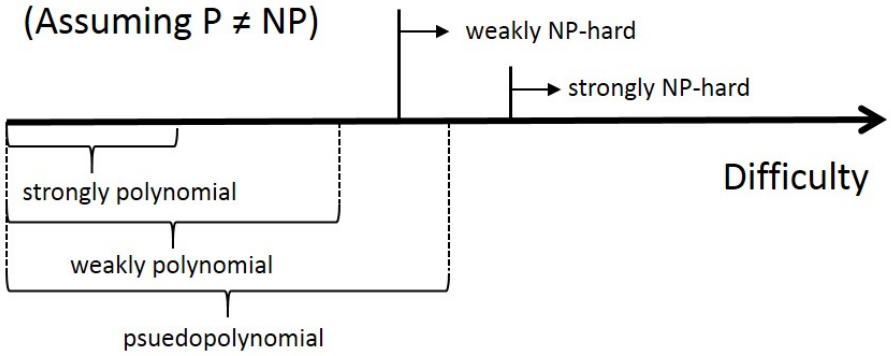
Correspondingly, the strength of a polynomial algorithm also depends on the encoding used. Suppose a_{max} is the largest number in the input.

- A **psuedopolynomial** algorithm is polynomial in n and a_{max}. (Unary encoding.) This often arises in dynamic programming, where the table

grows proportionally with the integer values. Examples: KNAPSACK, 2-PARTITION.

- A **weakly polynomial** algorithm is polynomial in n and $\log(a_{max})$. (Binary encoding.) This is the typical meaning of polynomial time.
- A **strongly polynomial** algorithm is polynomial in n alone.

The following diagram summarizes the relationships between these hardness types and algorithm strengths:



In practice, the integers involved in a problem are often only polynomially large in n, so a psuedopolynomial algorithm may be efficient. Thus there is practical significance in showing strong vs. weak NP-hardness, since only the former rules out psuedopolynomial algorithms.

3 Partition Problems

Consider the following problems:

- 2-PARTITION: Given integers $A = \{a_1, \ldots, a_n\}$, partition A into two sets A_1 and A_2 with equal sums. That is, such that $\sum_{x \in A_1} x = \sum_{x \in A_2} x = \frac{1}{2} \sum_{x \in A} x$. Also known simply as the PARTITION problem.
- 3-PARTITION: Given integers $A = \{a_1, \cdots, a_n\}$, partition A into $\frac{n}{3}$ sets (or "bins") $A_1 \ldots A_{n/3}$ of equal sums. That is, such that $\sum_{x \in A_1} x = \cdots = \sum_{x \in A_{n/3}} x = \frac{1}{n/3} \sum_{x \in A} x$.

2-PARTITION is *weakly* NP-Hard [4], while 3-PARTITION is *strongly* NP-Hard. In fact, it is strongly NP-Hard even under additional assumptions about A [2]. Namely, we may assume that all $a_i \in (t/4, t/2)$. This condition implies two things. First, each bin contains exactly 3 integers. Second, we may assume that all a_i are very close to $t/3$. This can be forced by adding some huge number (e.g. $a_{max} n^{100}$) to each a_i.

Both partition problems are useful in making hardness reductions, but only 3-PARTITION may be used to prove strong NP-hardness. Together with 3-SAT, 3-PARTITION may be the most important problem in proving NP-hardness.

4 Multiprocessor Scheduling

Given n jobs with processing times $a_1, a_2, \ldots, a_n$, and p sequential, identical processors, MULTIPROCESSOR-SCHEDULING is the problem of assigning jobs to processors in a way that minimizes the time from start until all jobs are completed (the makespan).

- **Reduction from** 2-PARTITION**.** Use the same a_i and let $p = 2$. The makespan clearly must be at least $t = \sum a_i/2$, and, in fact, such a makespan may be achieved only when there is a partition of A into two sets of size t. Thus, our reduction returns yes iff the minimum makespan is t. This implies MULTIPROCESSOR-SCHEDULING is weakly NP-hard.

In fact, for any constant p, MULTIPROCESSOR-SCHEDULING indeed admits a psuedopolynomial algorithm via dynamic programming on a table with constant dimensions. So the hardness result is tight for this case. However, in the following reduction, p is no longer constant, and we get a stronger hardness result.

- **Reduction from** 3-PARTITION**.** Use the same a_i and let $p = n/3$. By the same reasoning as above, we return yes iff the minimum makespan is $t = \sum a_i/(n/3)$. This implies MULTIPROCESSOR-SCHEDULING is strongly NP-hard. (In fact, this reduction was the motivation for Gary and Johnson to introduce 3-PARTITION in [2].)

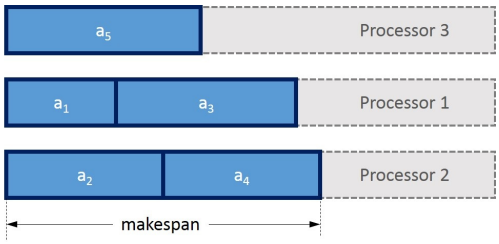


Figure 1: An assignment of processes to 3 processors.

Scribe: Thomas Pensyl
Lecture 3 **Date:** 09/09/2014

5 Rectangle Packing

Given set A of n rectangles, and a rectangle B, can we arrange all elements of A (via translation and rotation) into B without overlap? We call such an arrangement a *packing* of A into B. RECTANGLE PACKING asks whether such a packing exists.

- **Reduction from** 2-PARTITION**.** Let A be rectangles of size $(a_i \times \epsilon)$. Let $B = (t \times 2\epsilon)$, where $t = \frac{1}{2}\sum_i a_i$, and $\epsilon \ll 1$. This small value of ϵ rules out any useful rotation of the pieces. Then there is a valid 2-partition of $(a_1, \ldots, a_n)$ iff there is a valid packing of A into B. (Figure 2)
- **Reduction from** 3-PARTITION**.** This reduction is the same as above, except that we let $B = (t \times \frac{n}{3}\epsilon)$, where $t = \frac{1}{n/3}\sum_i a_i$, and again $\epsilon \ll 1$.

In the integer version of the problem, all rectangles have integer dimensions. To show strong NP-hardness for the integer version, we scale up the size of all rectangles. To reduce from 3-PARTITION, let A be rectangles of size $(na_i \times 1)$ and $B = (nt \times n/3)$ (recall we may assume that n in 3-PARTITION is a multiple of 3). Again, the rectangles are so long that rotation is ruled out in the solution. In fact, because we are using an exact packing, we need only rule out that a piece fits vertically in B, as any other rotation from the horizontal orientation would create space unfillable by any rectangles. Thus, it suffices to *add* $n/3$ to each a_i, that is, use $A_i = (\frac{n}{3} + a_i) \times 1$ and $B = (t + n) \times \frac{n}{3}$, greatly decreasing the size blowup.

This actually proves strong NP-hardness for EXACT PACKING, the special case of RECTANGLE PACKING in which the area of B is exactly the net area of all pieces in A. This is a stronger result, as it immediately implies the same hardness for the more general problem.

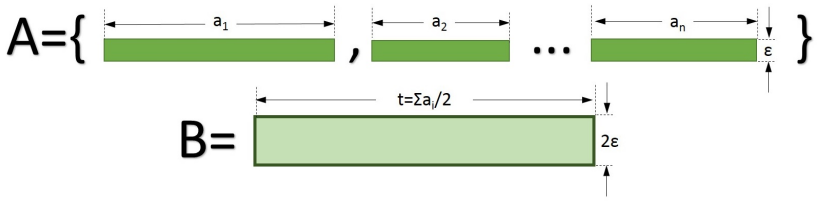


Figure 2: Here a packing of A into B is equivalent to a solution to 2-PARITITION on $\{a_1, \ldots, a_n\}$.

5.1 Square Packing

SQUARE PACKING is a special case of RECTANGLE PACKING where the pieces A and the bounding box B consist only of squares. This problem may also be reduced from 3-PARITITION[5], though the reduction is more involved.

First we describe a frame gadget. For some x, let A be one square of size $x(x+1)-1$, $x+2$ squares of size x, and x squares of size $x+1$, and B be a square of size $x(x+2)$. It may be shown that the "best" packing leaves only a rectangular gap of size $1 \times (x(x-1)-1)$. By scaling everything up, and adding additional dummy squares to fill part of the rectangle, we may force a rectangular gap of essentially arbitrary size.

- **Reduction from** 3-PARTITION**.** Create a frame gadget of height t and width $\frac{n}{3} \cdot \frac{t}{3} + \epsilon$, where $t = \frac{1}{n/3}\sum_i a_i$ and ϵ is very small. Then for each integer a_i, add a square of that size. Recall we may assume that all a_i are very close to $t/3$. Thus, the only possible packing is $\frac{n}{3}$ columns, each of 3 squares. For 3 squares to fit vertically, their sizes must sum to less than t. However, if no column has height greater than t, they all must be exactly t high. The ϵ allows some horizontal space for the small variations in widths of each column, but is not enough to allow for any significantly different configuration of the squares. We conclude there is a square packing iff there is a valid 3-partition.

6 Four Equivalent Puzzles

6.1 Edge-Matching

In EDGE-MATCHING, we are given a set A of unit square tiles with colored edges, and a target rectangle B, and ask if there exists a packing of A into B such that all tiles sharing an edge have matching colors.

Since all input A are of unit size, we cannot immediately reduce from PARTITION or 3-PARTITION. Instead, we will first construct gadgets composed of tiles that are forced to be joined together in a particular way. We first use unique, "unmatchable" colors to force tiles to the outer wall of B, thus creating a "frame". With the borders occupied, all remaining edges *must* be matched with another tile. Thus, we may use uniquely-colored pairs of edges to force tiles together, creating any composite shape we like.

- **Reduction from** 3-PARTITION. Create gadgets with dimension $a_i \times 1$, and a frame with inner dimension $t \times \frac{n}{3}$ (Figure 3). Furthermore, color the frame's inner horizontal edges and the pieces' outer horizontal edges a single color, and a use a second color for the vertical edges. This forces all pieces to be horizontally aligned, but does not otherwise restrict the arrangement of gadgets in the frame. Then there is a solution to 3-PARTITION iff there is a solution to this edge-matching puzzle.

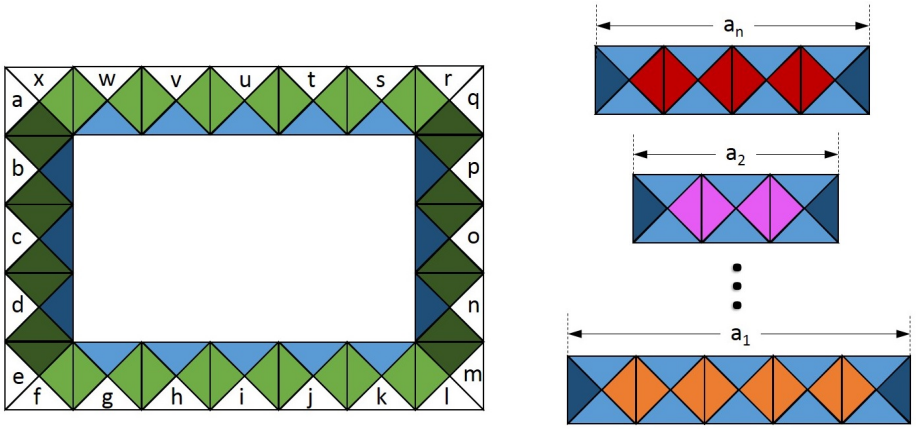


Figure 3: Unique colors a-x force the frame tiles to go on the outer edge of B, while the light and dark green edges put them on a particular side. The red/pink/orange edges force a single piece gadget together. The dark/light blue edges prevent the piece gadgets from rotating.

In the above reduction we create an instance of EDGE-MATCHING with $\Theta(\sum_i a_i)$ tiles. Thus, for the reduction to remain polynomial we need that each a_i is polynomial in the input size n. The strong NP-hardness of 3-PARTITION guarantees it is still NP-hard under this restriction. If we tried to reduce from 2-PARTITION it would not work because a_i could be exponential.

6.2 Signed Edge-Matching

In SIGNED EDGE-MATCHING, we again must pack unit tiles with colored edges into a rectangle B. However, now colors come in pairs: a&A, b&B, etc. A color may no longer be matched with itself, but instead must be matched with its complementary color.

- **Reduction from** EDGE-MATCHING**.** For each unsigned tile, create a 2×2 *supertile* of signed tiles as shown in Figure 4, and double the dimensions of B. We will claim there is a solution to the signed puzzle iff there is a solution to the unsigned puzzle, which proves the reduction.

  It is easy to see that given a solution to the unsigned puzzle, we may give a solution to the signed puzzle by arranging the supertiles in like manner. To show the converse, first observe that because of the unique color-pairs used internally, all tiles must form into their intended supertiles (or partial supertiles, if on the border of B.) Furthermore these supertiles must be aligned along a grid with spacing 2 (otherwise unfillable space is created). If the grid matches up with the borders, then all tiles belong to proper supertiles, and by replacing each supertile with the corresponding

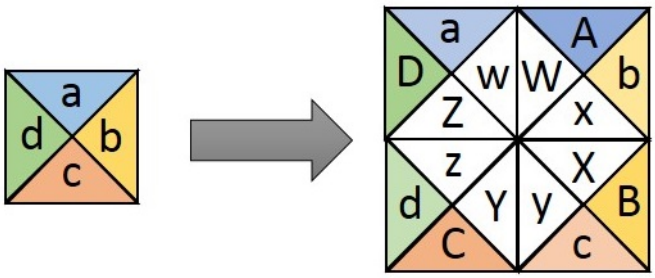


Figure 4: Creating a 'supertile' of signed tiles from an unsigned tile. The internal colors are unique to this supertile.

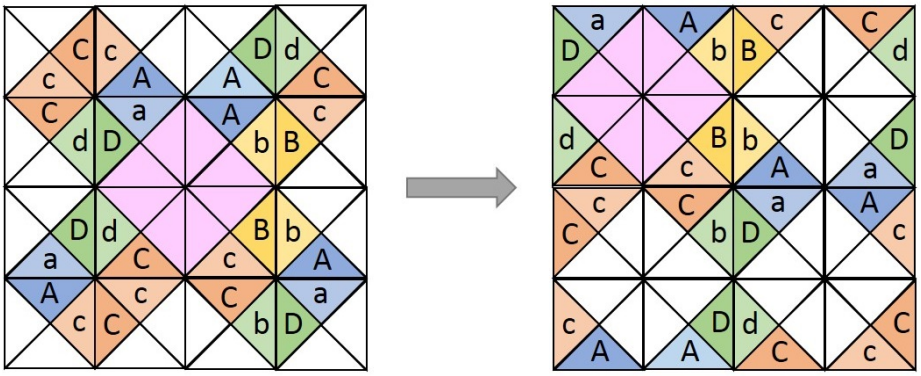


Figure 5: Aligning a mis-aligned solution by shifting left and up by 1. In this case, the supertile fragments directly across from each other line up. In general they may need to be rearranged, but are still guaranteed to fit.

unsigned tile, we get a valid solution to the unsigned puzzle.

If the grid does not match up with the borders, so that there are partial supertiles on the borders, then we may fix this it to by shifting all tiles up and/or left by 1 tile (Figure 5). All tiles now outside of B may be recombined with their proper supertiles (on the bottom and/or right rows) to form a proper grid. The external coloring of the supertiles implies that since the partial supertiles matched on the bottom and/or left, the completed supertiles must also match. Again, this easily translates to an unsigned solution, and the proof is complete.

6.3 Jigsaw Puzzles

Consider a classic jigsaw puzzle, except with no guiding picture, and ambiguous mates (i.e. there may be more than one piece that fits any particular tab or pocket). Such a jigsaw puzzle is very similar to a signed color-matching puzzle. In the jigsaw puzzle, instead of pairs of colors, each side of a piece has either a tab or a pocket. A tab only matches a pocket if it its shape is the exact inverse of the other. Also, some of the pieces have flat edges on 1 or 2 sides, which must be placed at the border.

Let SIGNED EDGE-MATCHING' be the problem of SIGNED EDGE-MATCHING on an $a \times b$ board where there are exactly $2(a + b)$ 'unmatchable' edges (i.e. it is obvious which edges must lie on the border). This special case may be seen to be NP-hard by digging into the previous reduction steps (from 3-PARTITION to UNSIGNED EDGE-MATCHING to SIGNED-EDGEMATCHING), as is shown in [1].

- **Reduction from** SIGNED EDGE-MATCHING'**.** We create a mapping from tiles to jigsaw pieces, by mapping the color of each edge to a jigsaw tab or pocket. First, map all 'unmatchable' colors to a flat edge: this forces these edges to lie on the border. Second, for every other pair of signed colors (c_i, C_i), let c_i map to a tab with shape s_i, and C_i map to a pocket of the same shape. Then two jigsaw edges match iff the signed colors match. Thus, configurations and solutions of the jigsaw puzzle map 1-to-1 with those of the signed edge-matching instance.

We also note that reverse mapping of edges gives a clean reduction from JIGSAW PUZZLE back to SIGNED EDGE-MATCHING.

6.4 Polyomino Packing

POLYOMINO PACKING is a generalization of EXACT RECTANGLE PACKING, where each piece is a *polyomino*, that is, an edge-to-edge joining of unit squares. For example, pieces from the game Tetris comprise all possible polyominos of area 4. We already know it is NP-hard because it contains RECTANGLE PACKING as a special case. However, the following reduction shows the same hardness even when all polyominos are "small".

- **Reduction from** JIGSAW**.** First map each type of tab/pocket to a unique binary string b of length l. Now create a polyomino for each jigsaw piece as follows (see Figure 6 for example). Start with a solid square of size $(4 + l) \times (4 + l)$. For jigsaw sides with tab type b, append unit squares along the polyomino side corresponding to the position of 1's in b, going in a clockwise direction. For pockets of type b, make unit square cuts into the polomino, going in a counter-clockwise direction. Leave each 2×2 square at the corner alone as padding. In this way, two polyomino edges fit together with no blank space iff their corresponding jigsaw edges fit together. Since this is exact packing, any blank space left between two polyominoes will be unfillable by our large polyominoes. Flat jigsaw edges

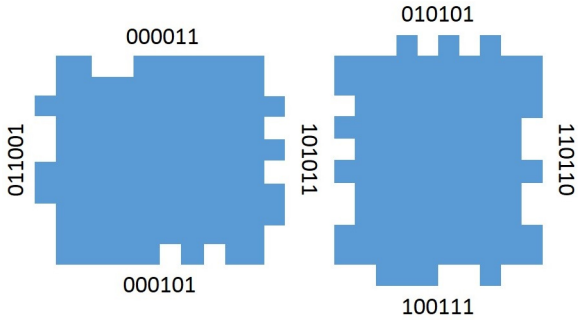


Figure 6: Encoding of jigsaw pieces into polyominos.

map to flat polyomino edges. Scaling the container dimensions up by $(4+l)$ completes the reduction. If n is the number of jigsaw pieces, then there are $O(n)$ types of tabs and pockets, so we need l to be $O(\log n)$. This shows it is NP-hard to decide EXACT POLYOMINO PACKING with pieces of dimension $O(\log n) \times O(\log n)$, and thus area $O(\log^2 n)$.

In fact, EXACT POLYOMINO packing remains NP-hard even when pieces have area $O(\log n)$. In the previous reduction, each edge has length $\Theta(l)$ and gets l 'bits' of information for encoding its jigsaw edge type. However, there are $\Theta(l^2)$ 'pixels' in the polyomino which comprise the solid center. What if we could instead give each edge $\Theta(l^2)$ bits to encode its edge type? In fact, Figures 7 and 8 show one potential polyomino design which accomplishes this. Now, using edge lengths of sizeonly $\Theta(\sqrt{\log n})$, we still get $\Theta(\log n)$ bits of information per edge: enough to encode the jigsaw edge type. This shows that EXACT POLYOMINO PACKING is NP-hard to decide for pieces of size $O(\sqrt{\log n} \times \sqrt{\log n})$ and area $O(\log n)$.

6.5 Closing the loop

The following is a slight modification of the reduction from 3-PARTITION to UNSIGNED EDGE-MATCHING.

- **Reduction from** EXACT POLYOMINO PACKING **to** UNSIGNED EDGE-MATCHING**.** As in Section 6.1, create an outer frame gadget and a gadget for each polyomino piece. Previously, the piece gadgets were $1 \times x$ rectangles. Now, we will create arbitrary polyominoes by fusing many tiles together in the same shape as the polyomino. To make sure the shape is preserved, use a unique color pair for each internal edge of the gadget. (This is the same method we used to force supertiles to stick together - see Figure 4). Lastly, in the gadgets in Figure 3, we used dark and light blue for both the internal edges of the frame, and the external edges of each piece, in order to keep the pieces from rotating. In polyominoes, we

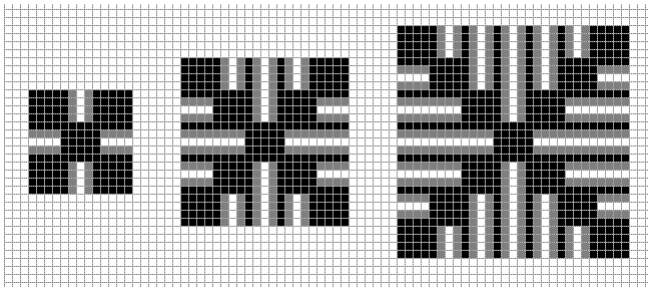


Figure 7: The grey area represents free 'bits' which may be used to encode the tab type. The black areas always remain solid, and guarantee contiguousness of the polyomino. (Likewise the white areas determine the shape of the 'key' and also are shaped to remain contiguous.) In the scheme depicted, a square with sidelength $8r + 5$ has $8r^2$ grey bits available for each edge.

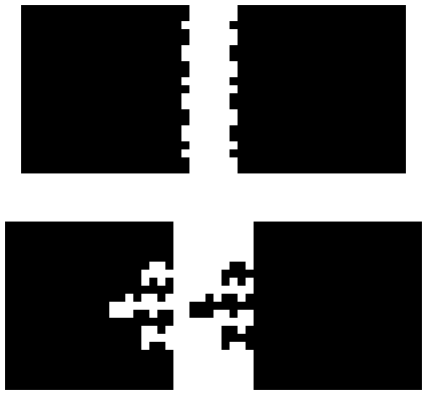


Figure 8: Using deeper 'keys' allows for more possible combinations than just using bumps and grooves on the edges.

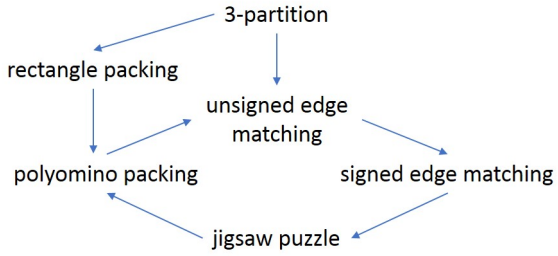


Figure 9: Each arrow represents a reduction from one problem to another.

want the pieces to be freely rotatable, so just use one color instead of 2 for these edges.

Thus, all four puzzle types are essentially reducible to one another and NP-complete, as shown in Figure 9. Furthermore, the blowup when converting between puzzle types is at worst logarithmic (as occurs when converting from JIGSAW to POLYAMINO PACKING).

References

[1] Demaine, Erik D., and Martin L. Demaine. "Jigsaw puzzle, edge matching, and polyomino packing: Connections and complexity." *Graphs and Combinatorics* 23.1 (2007): 195-208.

[2] Garey, Michael R., and David S. Johnson. "Complexity results for multiprocessor scheduling under resource constraints." *SIAM Journal on Computing* 4.4 (1975): 397-411.

[3] Hajiaghayi, Mohammad T. *Algorithmic Lower Bounds: Lecture 3– Handwritten notes*. 9 September 2014. University of Maryland.

[4] Karp, Richard M. *Reducibility among combinatorial problems.* Springer US, 1972.

[5] Leung, Joseph YT, et al. "Packing squares into a square." *Journal of Parallel and Distributed Computing* 10.3 (1990): 271-275.