Games, Puzzles, & Computation Class Notes

Written by: Angel, Austin, Kevin, and Lillian

Week 3: Jan 30th, 2017 - Feb 3rd, 2017

Review of the previous week's lab

"dots and boxes"

In the last lab session, one of the games that was demonstrated was "dots" (a game where players try to make boxes by connecting dots). One of the strategies of the game is called double-dealing, which is when a player gives up some of their boxes so that you can force the other player to make certain moves, leaving you with more boxes in the next turn.

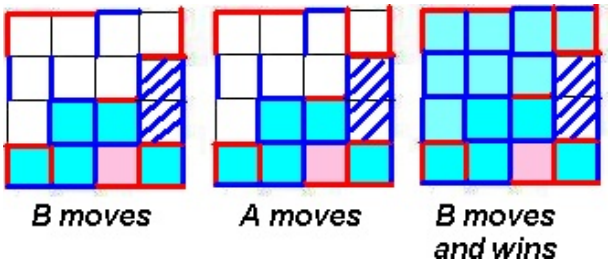

Figure 1: A simple example of B making a move to set themselves up for a win.

"hex"

This game has a large 1st player advantage. A good tip, when playing this game, is to pick blocks that will leave you with 2 other choices in the next round.
There can never be a tie in hex.

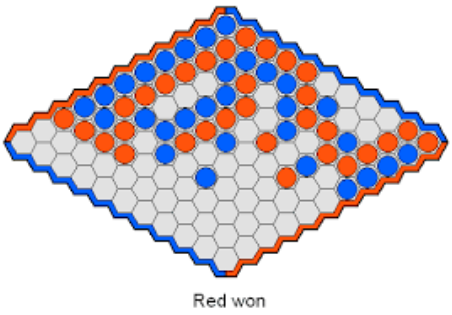

Figure 2: Red won because blue failed to block them.

There were 3 other games to play at the lab last week: Niya, Gobblet, and Quantum tic-tac-toe.

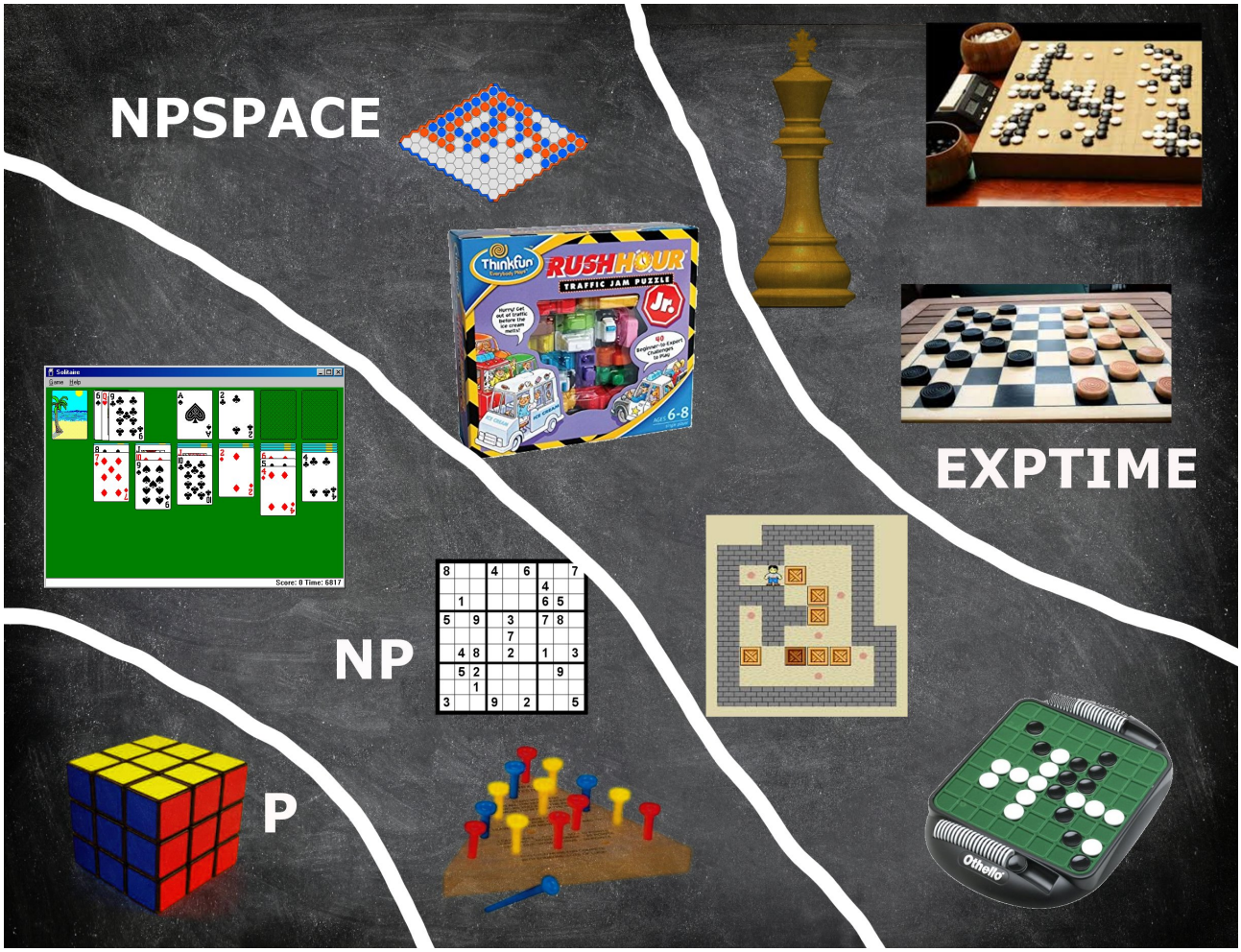

Figure 3: Examples of games and their complexities.

Classes of Complexity

Computation - Can be thought of as the process of producing an output from a set of inputs and a finite number of steps.

Turing Machine - In its simplest form, a Turing machine can be thought of as a computer with infinite memory.

Decision Problems - Problems whose computed output is either *yes* or *no*. All of the problems referred to in complexity classes are decision problems.

P - *Polynomial-Time*

- This class of decision problems contains all problems solvable in *polynomial* time by a Turing machine. $O(n^2)$

NP - *Nondeterministic Polynomial-Time*

- This class of decision problems contains all problems solvable by a "*nondeterministic* Turing machine" in polynomial time.
- Equivalently, if the answer to a problem is *yes*, then there is a *proof* of this that can be verified by a deterministic polynomial-time algorithm.

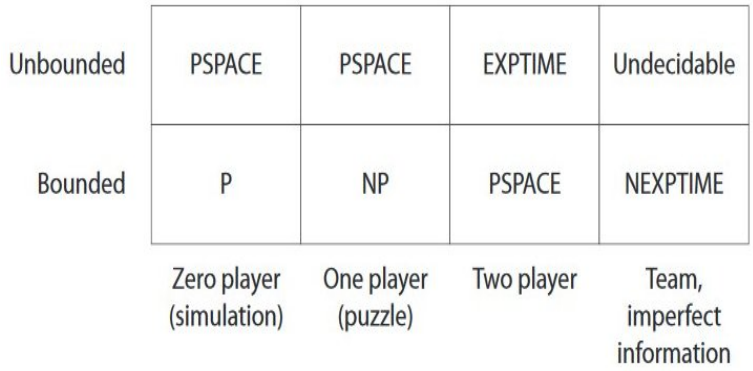

Figure 4: Examples of games and their complexities.

PSPACE *Polynomial-Space*

- This class of decision problems contains all problems are solvable by a Turing machine in polynomial space.

EXP - *Exponential-Time*

- This class of decision problems contains all problems in *exponential* time by a Turing machine. $O(2^n)$

R - *Recursive Languages*

- This class of decision problems all problems that are solvable by a Turing machine.
- This class is often identified with the class of "effectively computable" functions.

Other things to know...

co-NP - *compliment of NP*

- This class can be thought of as the opposite of NP. If the answer to a problem is *no*, then there is a *proof* of this that can be verified by a deterministic polynomial-time algorithm.

X-hard - (*NP*-hard, *EXP*-hard, etc...)

- A problem is X-hard for some complexity class X if it is at least as hard as the hardest problems in X. Proving a problem is X-hard is often done through reductions.

X-complete - (*NP*-complete, *EXP*-complete, etc...)

- A problem is X-complete for some complexity class X if it:
  1. Is in class X.
  2. Is X-hard.

Example of how to show a problem is in the class NP

To show that a problem is in the class NP, all that is needed is a polynomial time verification for an answer to that problem.
Problem: - Subset-Sum
Input: A set of integers S and an integer k
Output: Does there exist a set $T \subseteq S$ s.t. the sum of the elements of T is k?

Given: A set of integers S, an integer k, and an alleged "solution set" T:

1. Check that $T \subseteq S$
2. Check that $\sum_i t_i = k$
3. If 1 and 2 are true, output yes, otherwise, no.

Reductions

Reductions are algorithms that are used to transform one problem into another. Reduction from one problem to another problem is a way to show that the second problem is "at least as" difficult as the first. We say problem A is reducible to problem B if there exist an algorithm that solves problem B efficiently and can be used to solve problem A. Although there are many methods of algorithm reductions, we focus on a specific and very popular kind of reduction often used to prove NP-Completeness.

We use reductions in 2 different situations:
1.) When we are solving a problem that is similar to one we have solved, so we can use the solution from our original problem to solve the new one.
2.) When we are solving a problem that is similar to a difficult problem we have solved, so we obtain a proof by contradiction by arguing that the new problem is easy to solve, when it in fact is not.

Satisfiability (also known as Boolean Satisfiability Problem) is a decision problem of determining whether there exists a combination of literals (TRUE/FALSE) such that the expression is satisfied (OUTPUT = TRUE). If true, we call such an expression satisfiable (and similarly, unsatisfiable if the output is false).

Problem: - Satisfiability
Input: A set of Boolean Variables V and a set of clauses C over V
Output: Does there exist a satisfying truth assignment for C, i.e, a way to set the variables $V_1, \ldots, V_n$ true or false so that each clause contains at least one true literal?

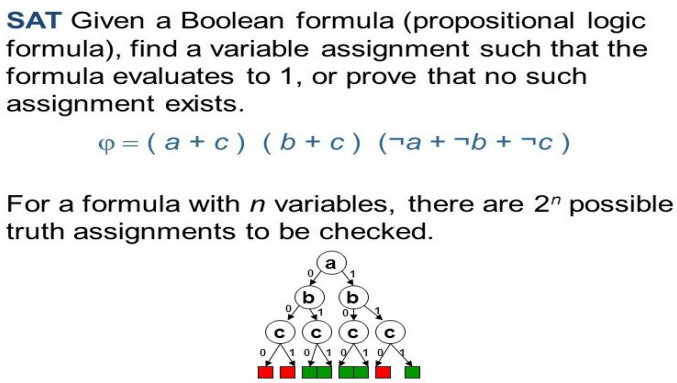

Figure 5: Adapted From 'The Quest for Efficient Boolean Satisfiability Solvers' - Sharad Malik

Literals are the Boolean Variables (either positive literal (x) or negative literal ($\neg x$)). A Clause can be described as the disjunction between two literals (X OR $\neg X$). When a formula is described as conjunctive normal form (CNF), it means the conjunction of various clauses ((x and not x) or (y or not y) or (z or not z)) .

3-Satisfiability It is easy to test Satisfiability on problems containing only two pairs of clauses, however, we are interested in problems containing of a larger class. How many literals per clause do you need to turn a problem from polynomial to hard?

Problem: - 3-Satisfiability (3-SAT)
Input: A collection of clauses C where each clause contains exactly 3 literals, over a set of Boolean Variables V.
Output: Is there a truth assignment to V such that each clause is satisfied?

Clique Problem
The clique problem is the computational problem of finding cliques (subsets of the vertices, all adjacent to each other, that form a complete subgraph). For a given graph $G = (V, E)$, the clique problem is to find whether G contains a clique of size $\geq k$.

Reducing 3-SAT to k-clique
Construct a graph G of k clusters with a maximum of 3 nodes in each cluster, and each node in a cluster is labeled with a literal from the clause. An edge is attached to all pairs of nodes in different clusters expect for the pair $(x, \neg x)$. Also, no edge is added between nodes of the same cluster.

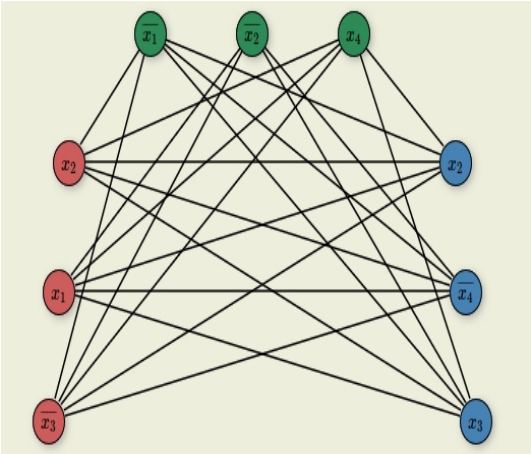

Figure 6: $b = (x_2 + x_1 + \neg x_3)(\neg x_1 + \neg x_2 + x_4)(x_2 + \neg x_4 + x_3)$

If two nodes in the graph are connected, the corresponding literals can be simultaneously be assigned True. (This is okay because there is on edge between the literals (X_i) and $(\neg X_i)$. If two literals, not from the same clause can be assigned True simultaneously, the nodes corresponding to these literals in the graph are connected. (Note that the construction of the graph takes polynomial time). We say G has a k-clique if and only if b is satisfiable. If b is satisfiable, let A be a satisfying assignment, and select from each clause a literal that is True in A to construct a set S. $|S| = k$. Since no two literals in A are from the same clause and they are simultaneously True, all the nodes are connected to each other in the graph, forming a k-clique. Hence, we say the graph has a k-clique.

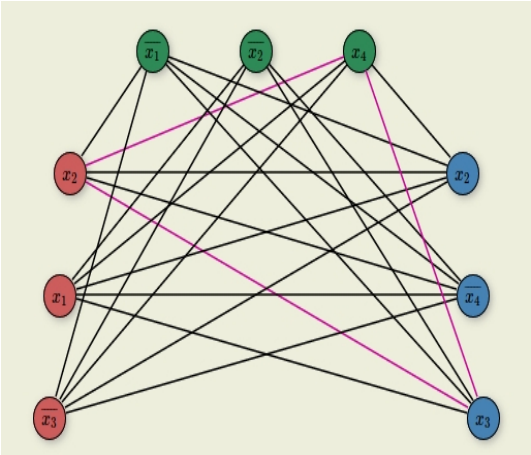

Figure 7: $b = (x_2 + x_1 + \neg x_3)(\neg x_1 + \neg x_2 + x_4)(x_2 + \neg x_4 + x_3)$

Good Reference Material

- Computational Complexity: A Modern Approach
  Princeton University
  pdf

- MIT OpenCourseWare - Video and Class Notes
  Lecture 23: Computational Complexity
  link

- University of Illinois - Class Notes
  Lecture 21: NP-Hard Problems
  pdf