

Lecture 20 Scribe Notes

Prof. Erik Demaine

1 Overview

This lecture completes our lectures on game characterization. Previously, we covered 0, 1, and 2 player games, and we focus on team games with imperfect information, corresponding to the last column in the game complexity chart in Figure 1, and 0-player games with a polynomial number of moves.

Team games with imperfect information fall into one of two categories – games that have a bounded number of moves can be shown to be in NEXPTIME, while games with an unbounded number of moves are undecidable. It is important to note that an unbounded number of moves can refer to not just exponential numbers of moves, but also an almost infinite number of moves.

There are no known "natural games" in either category, but bridge is team game with imperfect information and a bounded number of moves (all players must play their cards at a turn until there are no cards remaining), and is conjectured to be in NEXPTIME. Similarly, Rengo Kriegspiel (blind Go) is a team game with imperfect information and an unbounded number of moves and is conjectured to be undecidable.

Finally, we consider bounded 0 player games, such as Game of Life run for a polynomial number of moves. These simulations are in P, but also provide a notion of P-Completeness, and many games can be reduced to the Game of Life simulation. This is interesting because there is no way to parallelize a simulation of Game of Life.









unbounded				
	PSPACE	PSPACE	EXPTIME	Undecidable
bounded				
	P	NP	PSPACE	NEXPTIME
	0 players (simulation)	1 player (puzzle)	2 players (game)	team, imperfect info

Figure 1: A table showing the complexity classes for each type of gam, with bounded and unbounded referring to whether the number of moves is polynomially bounded or not.

2 Bounded Team Games

2.1 Dependency QBF (DQBF)

Dependency Quantified Boolean Formula (DQBF) was introduced by Peterson, Reif, and Azhar in [4].

This contrasts with QSAT, which is PSPACE-complete with n total variables, and the value of any variable may depend on all variables before it. However, the choice of Y_1 and Y_2 in DBQF depends only on the values of X_1 and X_2 , respectively. We note that X_1, X_2, Y_1, Y_2 can each represent n variables, but the values of these variables cannot all depend on each other.

Then the the game can be played with three players. The black player assigns the values of X_1 and X_2 . White player 1 assigns the values of Y_1 and white player 2 assigns the values of Y_2 . The private information in this game is the knowledge of X_1 and X_2 ; white player 1 only knows X_1 while white player 2 only knows X_2 . The private information distinguishes this game from a 2-player version. Then the associated question is: "Can white find an assignment of variables that satisfies the CNF formula?"

The problem belongs to NEXPTIME (In fact, it is NEXPTIME-Complete). This follows from the fact that there are only exponentially many choices for the value of X_1 and X_2 . Then given a set of values X_1 and X_2 , white player 1 and white player 2 may guess the winning strategy for the CNF formula, and verify that it is a solution if everyone plays optimally.

DBQF is a particularly interesting game, because it highlights that a bounded, private information team game played with one round can be as complex as a multi-round game. The white and black players only have one opportunity to set the values of the variable, but the game is still NEXPTIME-Complete. In all other game settings, alternation allows the players more power and makes games more complex.

2.2 Bounded Team Private Constraint Logic

Bounded Team Private Constraint Logic (TPCL) is a version of the Constraint Logic game played with three players, one black player and two white players, and a planar graph. The state of the edges in the game is private information, as each edge is marked so it is only visible to the white players, one of the white players, or the black player. Additionally, all edges are marked so they may be flipped only by the white team or only by the black team.

The same moves allowed in 2CL games apply, but players face the additional restriction that all moves that they make must be legal given visible information. Passes are also allowed, which makes it difficult to gain any information about changes to the state of the board based on the possible moves of other players. This is a necessary condition, because a black player may move an invisible edge, which may not change the available moves for a white player, thus giving them some knowledge about the black player's moves. By allowing passing, the white player does not know if the state of an invisible edge was changed, or if the black player decided to pass on a turn.

TPCL belongs to NEXPTIME. If there exists a winning strategy for the white player, the strategy is deterministic and is a function of the visible state. There are at most n different visible edges, so the number of possible states is exponential in the number of visible edges. Thus the white players

may guess a strategy at the beginning of the game, which takes exponentially many bits, and the white players play deterministically with that strategy to verify that it is a solution. The game may run for many rounds, but the information used to describe the moves is exponential, and the game belongs to NEXPTIME.

TPCL is also NEXPTIME-Complete, via a reduction from DBQF. We use the following gadgets to simulate the black player’s choice of values for X_1 and X_2 , and the white players choices for Y_1 and Y_2 . As in all CL games, we can only flip an edge once, and we can assign a value to a variable in DBQF only once.

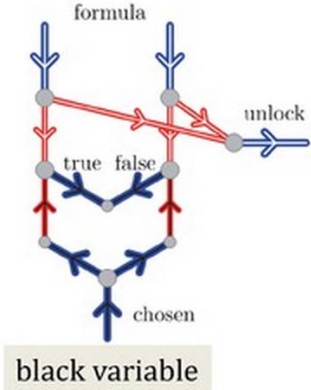


Figure 2: Black variable gadget. The solid edges are available for the black player to flip

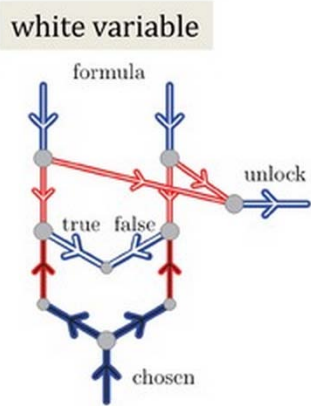


Figure 3: White variable gadget. The outlined edges are available for the white variable to flip

We allow black to first choose the value of the variables by flipping either the true or false edge in Figure 2, and both values of the variable cannot be chosen (both the true and false edge cannot be flipped) due to the vertex in the middle. Then white will flip the edge above the variable edge and black will flip the edge below the variable edge to (eventually) activate the formula and chosen edges respectively.

We see that all black variables are clustered in the figure below. There is an AND gadget connecting all the black variable gadgets via their chosen edge, forcing the black player to chose a value for each variable. Then there is a long wire to the final black edge, creating a threat line. Black wins

if they can flip this edge, but the path is exactly long enough so that if the white variables can satisfy the formula first, the white team will win. Therefore, they can only win the game if they assign a value to each of their variables. Most of the edges in the black variable gadget are visible to only white player 1 or 2. For example, if black is setting an X_1 variable, white player 2 will only be able to see the variable's chosen edge to know that the variable's value has been chosen, while white player 1 will know the state of several other edges.

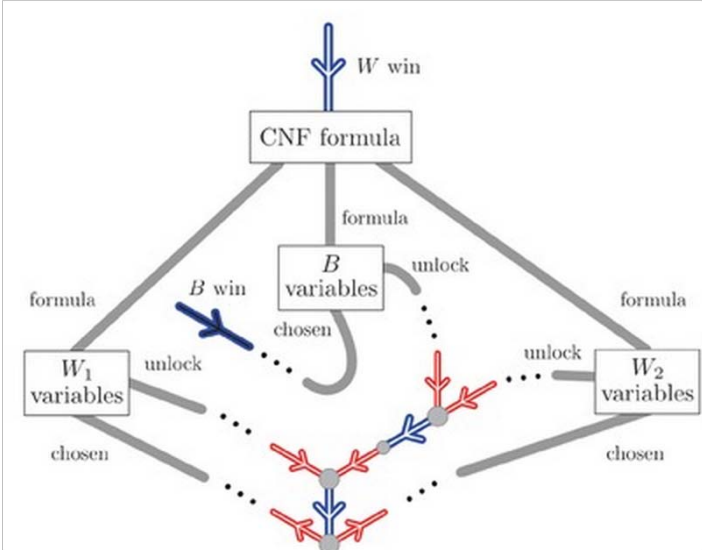


Figure 4: White variable gadget. The outlined edges are available for the white variable to flip

Until the black has chosen all variable values, white will be "twiddling their thumbs" and flip a useless edge back and forth. They can make other moves, but they will make these moves with less information about the black variable choices, which is suboptimal, as they have enough time to win even if they wait. Each of the white players will make moves based on the edges corresponding to the black variables that they can see.

Then white 1 and white two will set their variables. Their variable values will be combined via an "AND" of their chosen edges, forcing every variable to be assigned a value. Satisfying this AND allows an unlock of every variable, and allows the white player to begin flipping edges corresponding to the CNF formula. If the formula can be satisfied given the chosen variable values, white will have just enough time to flip the W win edge, and win the game. Thus we can reduce DQBF can be reduced to TPCL, and TPCL is NEXPTIME-Complete.

3 Unbounded Team Games and Undecidability

The fundamental problem of undecidability is the Halting Problem.

Definition 1. The Halting problem: *Given a Turing machine, does it ever halt?*

The undecidability of games is an interesting result because Turing machine run for an arbitrary

amount of time and have an arbitrary amount of space to store information and their state. However, games have limited board space and cannot represent the state of an arbitrarily large Turing machine.

3.1 Unbounded Team Computation Game

Now, we formally define an Unbounded Team Computation Game as defined in [2]. An instance of a team computation game is a space- k algorithm or Turing machine. The algorithm begins with blank memory, and the Turing machine begins with a blank tape. Game play proceeds as follows:

- There are two white players and one black player.
- The black player is required to run the algorithm/machine for k additional time steps.
- The black player is required to run the algorithm/machine for k additional time steps.
- The algorithm may return the fact that black wins, white wins and determine the winner.
- If the algorithm does not return an output, the black player sets $x_1, x_2 \in \{A, B\}$
- Player White i sees only x_i and can set the value at memory position m_i . They know k , but do not know the state of the algorithm/machine. Their only possible move is to set the value of m_i , which is one cell of memory/tape, but this cell may hold more than 1 bit.

The associated question is: "Does the white team have a forced win?" We show this problem is undecidable via a reduction from the Halting Problem.

The reduction proceeds as follows. Given a Turing machine, we build an $O(1)$ -space algorithm to check that the white players' symbols m_i provide a valid computation history trace of the form $\#state_0 \#state_1 \dots \#haltstate$. The state includes information about the Turing machine's tape and the instruction it is currently processing, and the $\#$ is used to separate information about the different states.

This is difficult because we have only constant space to verify that $state_{i+1}$ follows from $state_i$, and in fact we are not guaranteed to be able to store all information about a previous state in memory. Then instead both white players have two pointers A, B into the common history. Then if $x_i = A$, player i must request character at A and advance A , where the black player sets x_i equal to A or B , as above.

Then if white has a winning strategy, white must win no matter black's strategy, but black plays nondeterministically, requesting A and B randomly.

The algorithm checks that White 1's x_1 state = White 2's x_2 state if black has been making moves (x_1, x_2) moves since the last $\#$, which guarantees that the pointers are at the same place throughout execution and ensures that they have identical states. Then if we make (x_1, \bar{x}_2) moves until White 1 reports $\#$, which will cause $W1$ and $W2$ will be out of sync by one. Then we run (x_1, x_2) moves to check that this is a valid transition from $W2 \rightarrow W1$. If the algorithm finds an invalid transition, then the algorithm reports that black wins.

Thus we can see that the white player must be simulating the Turing machine to report the valid computation history, since the black player may choose any two values to play. Only if white has the valid values are they guaranteed to win regardless of black's moves.

Thus we have a reduction from the Halting Problem to Unbounded Team Computation Game, which is therefore also undecidable.

3.2 Team Formula Game and Unbounded TPCL

In Team Formula Game, the black player sets X such that $F(X, X', Y_1, Y_2)$, and $F'(X, X')$ are true. The black player wins if $G(X)$ is true. However, white 1 sets Y_1 , seeing only X_1 and white 2 sets Y_2 seeing only X_2 , where X_1 and X_2 are some values in X . This problem is also undecidable by a reduction from the unbounded Team Computation Game, where we can simulate the Turing machine as a formula.

This can further be used to show that (unbounded) TPCL is also undecidable, via the following gadget. The white players set the values of Y_1 and Y_2 , and these values are fanned into the circuit if they satisfy F and F' . Then the white team wins if it can flip its target edge, and black wins if it can flip its target edge. Thus TPCL is undecidable for three players, and gadgets can be used to show that TPCL with three players is undecidable even on a planar graph.

4 Parallelism and P-Completeness

Typically, we use weak models to show that computation is possible even in the worst conditions. However, to show give upper bounds, we use strong models to demonstrate the impossibility of some results. Therefore, the result is definitely impossible with more natural or weaker computational models. This motivates our discussion of NC (Nick's Class)

4.1 NC

NC or Nick's Class is named after Nick Pippinger, and describes the set of problems that are solvable in $\log^{O(1)} n$ time using $n^{O(1)}$ processors. Alternatively, we consider problems that can be solved by a circuit of size $n^{O(1)}$ and depth $\log^{O(1)} n$.

A problem that falls in NC is sorting n numbers. A simple (but inefficient) algorithm can do this with n^2 processors, each of which is assigned to compute a comparison between a pair of items. We can store the results of these computations in a matrix. And for each element i , we sum the number of items less than it by using a binary tree to sum the corresponding values in the column of the matrix in $O(\log n)$ time. We can then place every element in the sorted list according to its sum, which is equivalent to the number of numbers smaller than it in the list.

Now, NC is contained in P, since with polynomial parallelism, the algorithm is solvable in poly-log time. Therefore, even with out parallelism, the algorithm should be solvable in polynomial time.

4.2 P-Hard

We now describe P-Hard problems.

Definition 2. *A problem is P-Hard if all problems $\in P$ can be reduced to the problem via an NC-reduction.*

This implies that P-Hard problems are not in NC, assuming $NC \neq P$. We can also define P-Complete problems as problems in P that are P-Hard.

4.3 P-Complete problems

We discuss several P-Complete problems.

The first problem is General Machine Simulation: Given a sequential algorithm and a time bound t , does the algorithm say "YES" within t steps? Now, t must be encoded in unary to ensure that it is not exponential in the size of the machine, which would force the algorithm to run for an exponential amount of time, and the problem would be in EXP.

It is relatively straightforward to show that all problems in P can be reduced to this problem. If a problem is in P, there exists a sequential algorithm to solve it in polynomial time, and we can set t equal to that amount of time, and run this problem.

The second is the Circuit Value Problem (CVP), as defined by [3], which is similar to Circuit-SAT. Given an acyclic Boolean circuit and input bits, is the output true? The problem can be solved in linear time by sequentially simulating the circuit. However, it generally cannot be parallelized because a gate at a greater depth cannot be simulated until its inputs are ready.

Several other CVP problems are also P-Complete.

- NAND CVP – uses only NAND gates
- NOR CVP
- Monotone CVP – uses only AND/OR gates
- Alternating Monotone CVP (AMCVP) – alternates AND/OR down every path, starting and ending with OR
- Fanin-2, Fanout-2 (AM2CVP) – all gate have in and out degree 2, but we allow outputs other than the one of interest
- Synchronous AM2CVP – all inputs to each gate have same depth, useful for constraint logic
- Planar CVP

However, we note that Planar Monotone CVP is in NC.

Then, beginning from the assumption that Monotone CVP is P-Hard, we show that the lower items on the list are also P-Hard [1]. First, we show that it is P-Hard to solve the problem even if our circuit begins and ends with an OR, via the following gadgets. To begin with an OR, simply OR

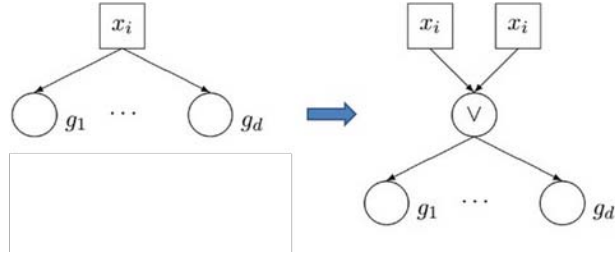


Figure 5: Gadget for an starting OR

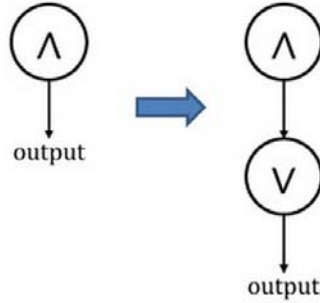


Figure 6: Gadget for ending OR

every input with itself and use the output of the OR as input to the gates. To end with an OR (if one is not already used), use the output of the AND gate as the input to a 1 input OR gate.

Next, we show that $\text{fanout} \leq 2$ is P-Hard via the gadget in Figure 7. If we have fanout greater than 2, we simply create a binary tree of OR gates, with fanout at most 2 and fanin at least 1.

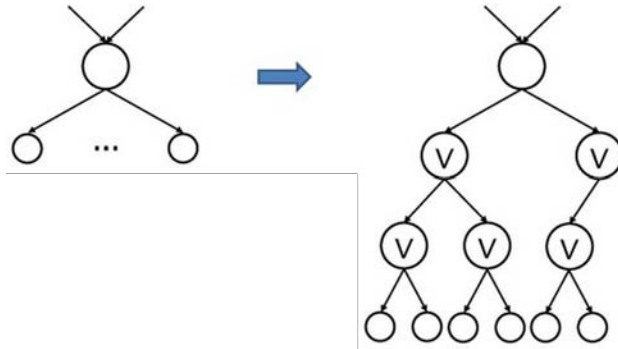


Figure 7: Gadget for fanout at most 2

Then we show alternating is also P-Hard with the gadget in Figure 8.

Then we show that Fanin-Exactly-2 is also P-Hard by converting Fanin ≥ 2 and Fanin-1 to Fanin-2 via the following gadgets.

Next, we show that Fanout-2 is preserved by making two copies of our circuit and merging the copies as depicted in Figure 10. We note that x'' , y_1 , and y_2 each have a single output, and we allow these gates to be a part of the "multiple" outputs, though these outputs that not the output

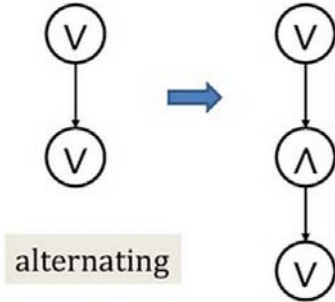


Figure 8: Gadget to allow gate alternation

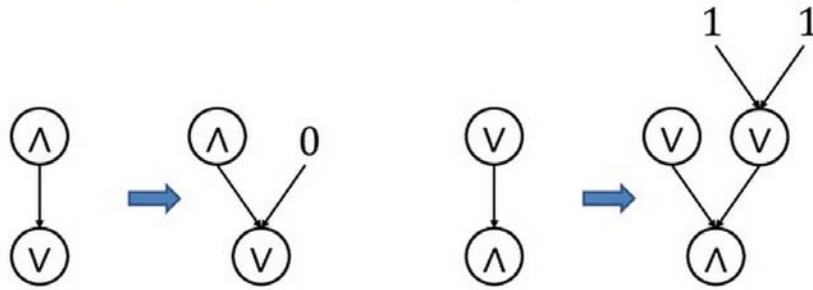


Figure 9: Gadget to ensure all gates have Fanin-2

we are specifically concerned with.

Next we show that Synchronous CVP is also P-Hard. We make $\frac{n}{2}$ copies of the circuit, where the circuit has n gates. The i^{th} copy will contain gates at depth $2i$ and $2i + 1$, where depth $2i$ gates are AND gates and inputs, while $2i + 1$ gates are ORs. The outputs for the ANDs are fed to the inputs of the ORs in the same copy, while the outputs of the ORs are inputs to the next copy.

However, we wish to ensure that the inputs to copy i arrive precisely when the other values are sent to copy i , so we delay them while maintaining Fanin-2, Fanout-2, and alternation, using the gadget on the left side of Figure 11.

We also consider Bounded Deterministic Constraint Logic, which is also P-Hard, via a reduction from Synchronous CVP. As usual, we are only allowed to change the orientation of an edge at most once. A diagram visualizing the reduction is given in Figure 12, and flipping the last edge is equivalent to the circuit outputting true.

4.4 Lexically first Maximal Independent Set

Finally, we consider the problem of finding a maximal independent set. While this problem is NP-Hard, we consider the problem of finding the lexically first maximal independent set, which is in P, by using the following polynomial-time algorithm.

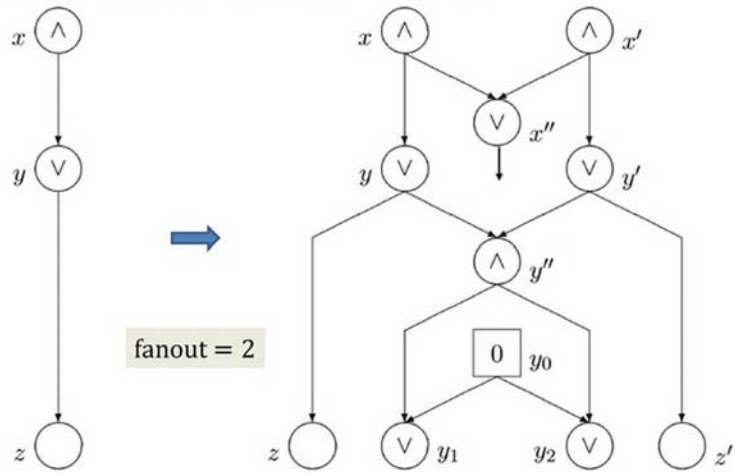


Figure 10: Gadget to ensure all gates have Fanout-exactly-2

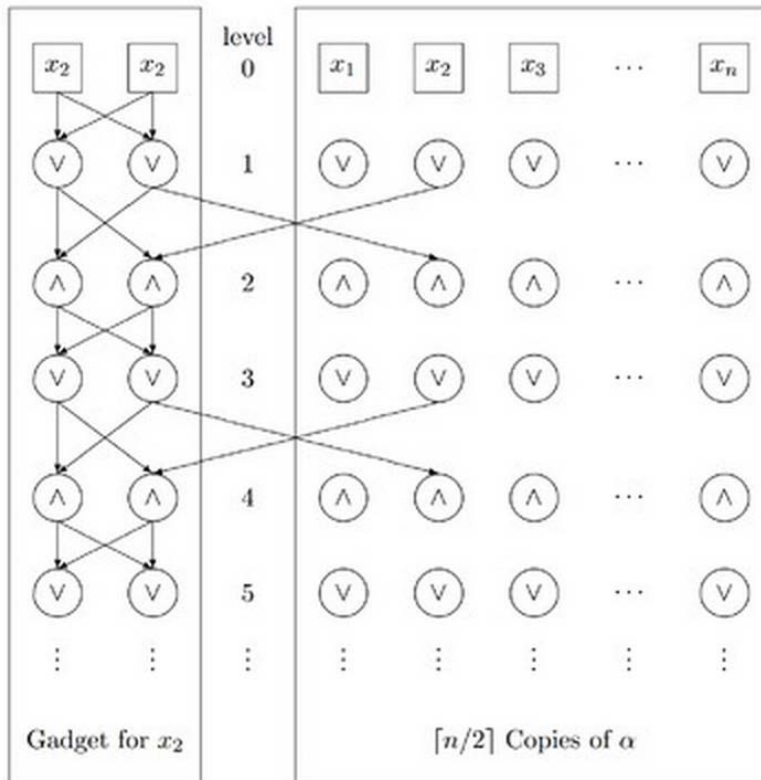


Figure 11: Gadget for Synchronous CVP

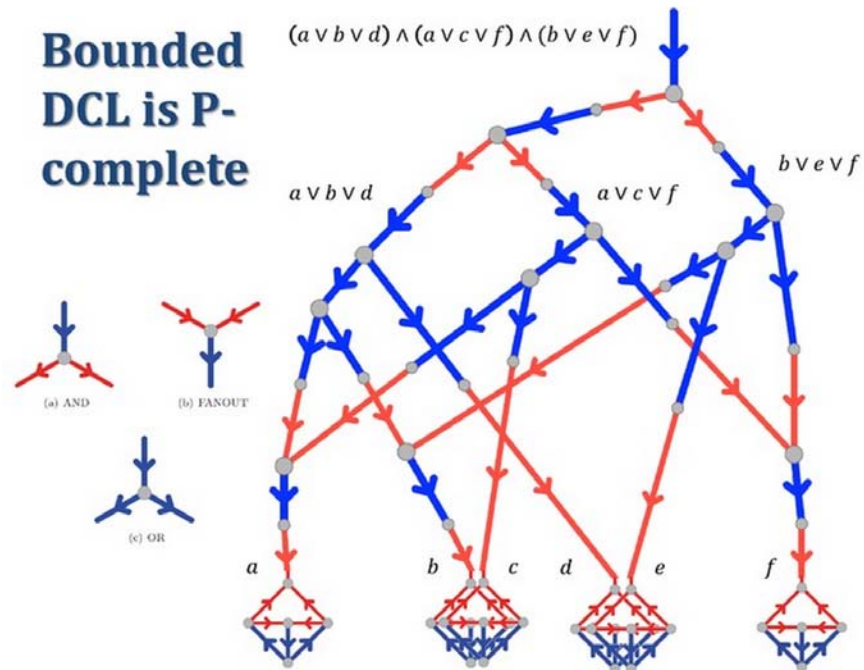


Figure 12: Reduction for Bounded DCL

Lexically First Maximal Independent Set:

$S = \emptyset$

for $i := 1, 2 \dots V$

 if v not adjacent to S :

$S = S \cup \{v\}$

 end

end

return S

The problem of finding the Lexically First Maximal Independent Set is P-Complete via a reduction from NOR CVP. First, we note that topological sorting can be done in NC. Then we build a graph based on the circuit. The first vertex is labelled the vertex, which is connected to all of the 0 inputs, which are also converted into nodes. We also replace every gate with a node. Then the vertices are ordered in a topological sort, where the null vertex has the lowest number, then the 0 inputs, etc. Then the last vertex V will be in S , if and only if the original circuit outputs a 1, via induction.

4.5 More P-Complete problems

Finally, we list a few additional P-Complete problems, and some open problems conjectured to be P-Complete

- Bounded Game of Life

- 1-Dimensional Cellular Automata
- Acyclic Generalized Geography
- Is a point p on the k^{th} layer of the convex hull of a point set?
- Multi-list ranking – Given an item, what is its rank in the union of lists?
- Does $a \bmod b_1 \bmod b_2 \dots \bmod b_n = 0$?
- Linear Programming with 0-1 coefficients (also strongly P-Complete)
- Max-flow

Open problems:

- Are two numbers relatively prime?
- Compute $a^b \bmod c$
- Maximum matching with large edge weights
- Graph isomorphism with bounded degree

References

- [1] Raymond Greenlaw, James Hoover, and Walter Ruzzo. *Limits to Parallel Computation: P-Completeness Theory*. Oxford University Press, New York, 1995.
- [2] Robert A. Hearn and Erik D. Demaine. *Game, Puzzles, and Computation*. A K Peters/CRC Press, 1st edition, July 2009.
- [3] Richard E. Ladner. The circuit value problem is log space complete for p. *SIGACT News*, 7(1):18–20, January 1975.
- [4] G. Peterson, J. Reif, and S. Azhar. Lower bounds for multiplayer noncooperative games of incomplete information. *Computers & Mathematics with Applications*, 41(78):957 – 992, 2001.

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>.