CS120: Intro. to Algorithms and their Limitations Anshu & Hesterberg

Lecture 23: Complexity of Games and Puzzles

Harvard SEAS - Fall 2023 *2023-11-28*

1 Announcements

2 Loose end from last time: unsolvability of 3Coloring

Our next example of an unsolvable problem is the following:

Input	: A RAM program P
Output	: `yes` if P correctly solves the graph 3-coloring problem, `no` otherwise

Computational Problem Solves3Coloring

Theorem 2.1. *Solves3Coloring is unsolvable.*

Proof.

□

3 Jigsaw puzzle: modeling

Definition 3.1. A *jigsaw puzzle* consists of natural numbers m and n, called its *dimensions*, and a set $S = \{t_0, t_1, \ldots, t_{mn-1}\}$ of "jigsaw tiles".

A *jigsaw tile* consists of four integers t.left, t.right, t.top, and t.bottom.

A *jigsaw partial solution* is a function $f : S \to ([m] \times [n]) \cup \{\text{"unassigned"}\}$ such that:

1. For all $t, t' \in S$, $f(t) \neq f(t')$ or $f(t) =$ "unassigned".
2. For all $t, t' \in S$, if $f(t) = f(t') + (0, 1)$, then f(t').top = -f(t).bottom.
3. For all $t, t' \in S$, if $f(t) = f(t') + (1, 0)$, then f(t').right = -f(t).left.

A *jigsaw solution* is a jigsaw partial solution in which no piece is "unassigned".

Question: What aspects of real-life jigsaw puzzles have we not modeled with the definitions above?

4 One-player jigsaws

We can state the problem of finding a solution to a jigsaw puzzle as a computational problem:

Input	**:** A jigsaw puzzle, as defined above.
Output	**:** A jigsaw solution, as defined above.

Computational Problem JigsawA1

One can think of solving a jigsaw puzzle as playing a 1-player game where the moves are placing jigsaw pieces. The "1" in the name of the computational problem reflects the one-player game.

Theorem 4.1. *JigsawA1 is* $\mathsf{NP}_{\mathsf{search}}$*-complete.*

Proof. We need to check that JigsawA1 is in $\mathsf{NP}_{\mathsf{search}}$ and that it's $\mathsf{NP}_{\mathsf{search}}$-hard.

JigsawA1 is in $\mathsf{NP}_{\mathsf{search}}$: given a proposed jigsaw solution, we can verify that it's correct by checking the compatibility of each pair of adjacent pieces: that is, check the condition defining a jigsaw partial solution.

JigsawA1 is $\mathsf{NP}_{\mathsf{search}}$-hard. We'll prove so by reduction from LongPath, which you proved was $\mathsf{NP}_{\mathsf{search}}$-hard in SRE4. Recall the definition of LongPath:

Input	**:** A digraph $G = (V, E)$, two vertices $s, t \in V$, and a path-length $k \in \mathbb{N}$
Output	**:** A path from s to t in G of length k, if one exists

Computational Problem LongPath

In fact, LongPath is $\mathsf{NP}_{\mathsf{search}}$-hard even if $k = n - 1$, so we'll reduce from that problem. We'll also assume that no edges leave t, since those edges can't be used in any path ending at t.

Given an input to LongPath, we'll construct a jigsaw puzzle whose dimensions are $(n + 2m + 4) \times (n + 2m + 4)$. In particular, we'll choose tiles that make two gadgets: $(n + 2m + 4)^2 - m$ tiles which make a framework gadget, and m tiles which make a path-and-disposal gadget.

Framework gadget:

Path-and-disposal gadget:

Proof of correctness:

□

5 Two-player jigsaws

We could define a two-player game version of jigsaw-solving: the players take turns placing a tile (starting with t_0, then t_1, etc.) adjacent to some already-assigned tile. Player 0 wants to solve the jigsaw puzzle and player 1 wants to prevent it from being solved.

Input	**:** A jigsaw puzzle
Output	**:** True if player 0 has a strategy in the two-player game version of jigsaw-solving which guarantees a win (i.e. a solved puzzle); false otherwise.

Computational Problem JigsawA2

JigsawA2 is complete for PSPACE, a complexity class we haven't defined yet:

Definition 5.1. PSPACE is the set of decision problems which can be solved in polynomial space. That is, PSPACE is the set of decision problems solved by some Word-RAM algorithm which, on inputs of size n, calls MALLOC at most $O(n^c)$ times, for some constant c.

Proving that JigsawA2 is PSPACE-hard is out of scope for CS 120, but we can prove that JigsawA2 is *in* PSPACE:

Theorem 5.2. *JigsawA2 is in PSPACE.*

Proof. A complete two-player jigsaw game lasts (at most) mn turns. For each $i \in [mn]$, we'll store in the ith cell $M[i]$ of memory, a move (i.e. a valid piece placement) that could be made in the ith turn.

Algorithm:

Proof of Correctness:

Efficiency: □

6 QSAT and 2-player bounded games.

Almost nothing in our proof that JigsawA2 is in PSPACE depended on the details of what JigsawA2 is. In fact, the only facts we used were that:

1. If we know the complete history of the game, we can tell whether player 0 won.
2. We can list all (polynomially many) possibilities for each move.
3. We have a polynomial bound on the length of the game.

So, the same argument proves that *every* game with the same properties is in PSPACE. For instance, we can turn SAT into a game by letting the players take turns choosing a value for each variable (in order), where player 0 wins if the SAT formula ends up satisfied. Deciding whether player 0 has a winning strategy in that game is called "Quantified Satisfiability" (QSAT)[1], and the same argument shows that it's in PSPACE.

Input	**:** A boolean formula φ on n variables $x_0, x_1, \ldots, x_{n-1}$
Output	**:** True iff $\exists x_0 \forall x_1 \forall x_2 \exists x_3 \cdots : \varphi(x_0, \ldots, x_{n-1})$. False otherwise.

Computational Problem QuantifiedSatisfiability

QSAT is the canonical example of a PSPACE-complete problem, although proving that QSAT is PSPACE-hard is out of the scope of CS 120.

As another example, those properties are true of chess, even if we generalize it to an $n \times n$ board:

1. If we know the complete history of the game, we can reconstruct the final position and see whether player 1's king is checkmated.
2. The board has only polynomially many pieces on it, each with polynomially many possible moves.
3. Chess has a polynomial bound on the number of moves, because if 50 moves pass without any pawns moving or pieces being captured, the game ends. Pawns can be advanced only polynomially many times before they all reach the end of the board, and pieces can be captured at most polynomially many times before there are none left to capture, making the length of the game at most polynomial.

In fact, deciding whether white wins generalized chess (with that 50-move rule) is PSPACE-hard as well, making it PSPACE-complete.

7 Unbounded-time games

Our proof that chess is in PSPACE relied on the 50-move rule. That rule was created in the belief that it would never end games that it's possible for one player to win, only save time by ending games where neither player could make progress. However, in the last few decades, it's been proved that there are chess positions which one player could win without the 50-move rule, but can't win before the 50-move rule ends the game. What if we got rid of that rule?

[1] QSAT was historically called "True Quantified Boolean Formula" (TQBF).

Even without the 50-move rule, we need only polynomially much memory to describe the state of a chess board, so one might hope that solving the game would still be in PSPACE. Our previous algorithm can no longer get by with polynomially much space, since it stores an entire history of the game, which might be of more than polynomial size.

In fact, chess without the 50-move rule turns out to be EXP-complete. Modern computer scientists believe (but have not proven) that there are problems solvable in exponential time but not solvable in polynomial space, so chess without the 50-move rule is probably not in PSPACE.

We could similarly make a variant of the 2-player jigsaw game without a polynomial bound on the length of the game: for instance, say that on a player's turn, a player can *either* place their next piece adjacent to a previously one *or move one of their previously-placed pieces* to an adjacent legal position. That game, called JigsawB2, is also in EXP, and is probably EXP-complete (although, as far as the instructors are aware, no one has worked through the details to prove so).

8 Unbounded-space games

There's one final natural way we could extend the problem of jigsaw puzzles: make the puzzle itself infinite. That is, define an infinite jigsaw as follows:

Definition 8.1. An *infinite jigsaw puzzle* consists of a set $S = \{t_0, t_1, \ldots\}$ of "jigsaw tiles".

A *jigsaw tile* consists of four integers t.left, t.right, t.top, and t.bottom.

A *jigsaw partial solution* is a function $f : S \to (\mathbb{Z} \times \mathbb{Z}) \cup \{\text{"unassigned"}\}$ such that:

1. For all $t, t' \in S$, $f(t) \neq f(t')$ or $f(t) =$ "unassigned".
2. For all $t, t' \in S$, if $f(t) = f(t') + (0, 1)$, then f(t).top = -f(t').bottom.
3. For all $t, t' \in S$, if $f(t) = f(t') + (1, 0)$, then f(t).right = -f(t').left.

A *jigsaw solution* is a jigsaw partial solution in which no piece is "unassigned".

Input	**:** A jigsaw puzzle, as defined above.
Output	**:** A jigsaw solution, as defined above.

Computational Problem JigsawC1

This problem is undecidable, but we won't have time to prove it in class.