Introduction to Part II

Created	@Nov 23, 2020 3:56 PM
Tags	Playing Games for Algo

Introduction

In Part-I we have learnt about Computation, Universal Turing Machines and Complexity Classes. We have seen how powerful the idea of Reduction can be in classifying "problems" in "certain classes" and have demonstrated the same in the shrewd proof stating that it is NP-Complete to win a generalized level in the Game Celeste.

Now in this part, we shall talk further about Games - how to model them, classify them, provided generalized theorems and discuss how and why having knowledge about such aspects of different games is important not only to our understanding of Complexity Theory but also to the world of Computation, in general.

Why Games?

Games involve the most vibrant (and in many cases even the oldest) set of computational problems. Chess, for example, is an ancient game believed to have originated in India around 1500 years ago and has been proven to be EXPTIME-Complete (in exclusion of the fifty-move rule) in 1981.

Games serve as models of computation which have been used quite prevalently used to mathematically model real-life scenarios. For example, classical game theory deals with several games involving rational decision making strategies and has found applications in computer science, biology and social sciences.

Games not only serve as a means of understanding computation but also decision making and behavioral relations. This is why they have been often found associated with numerous breakthroughs in artificial intelligence. It is quite astonishing as how games often are found to be embedded in deep computational problems and yet require incredibly less to none formal understanding to be played.

Moreover, puzzles and simulations can often be helpful to encapsulate real life problems especially in fields like bioinformatics.

Thus, it would indeed be surprising if understanding of games and augmented reality would provide us with no further **help in the understanding of the nature** as well as in **trying to answer some of the major philosophical questions** encompassing life and reality.

Game Theory and Complexity

Created	@Nov 18, 2020 8:05 AM
Tags	Playing Games for Algo

Game Theory

Game Theory serves as a good formalism to study a certain category of games (mostly non-cooperative multi-player games). It is very well studied field and we shall start of with it and then move forward to other formalisms and study more categories of games like simulations, puzzles, their video game counterparts as well as multi-player video games.

Introduction

> An equilibrium is not always an optimum; it might not even be good.

One very important common motivation of both Game Theory and Computer Science is to be able to **model rationality** and solve cognitive tasks such as **negotiation**.

In this module, we shall introduce Game Theory with a Complexity Theory minded approach. As it turns out, Game Theory has given rise to several interesting challenges with respect to computational . complexity especially in the realm of the complexity classes PPAD and PLS.

Prisoner's Dilemma and Nash Equilibrium

In the movie 'A Beautiful Mind', there is a memorable scene where we find Russell Crowe playing Dr. John Nash say that Adam Smith's statement - "In competition, individual ambition serves the common good" - was incomplete. Instead, Nash postulates that the best result is for everyone in the group doing what's best for himself and the group. This conversation serves as a very informal introduction to the idea behind Nash Equilibrium. Here, we shall now formalise it starting with an example.

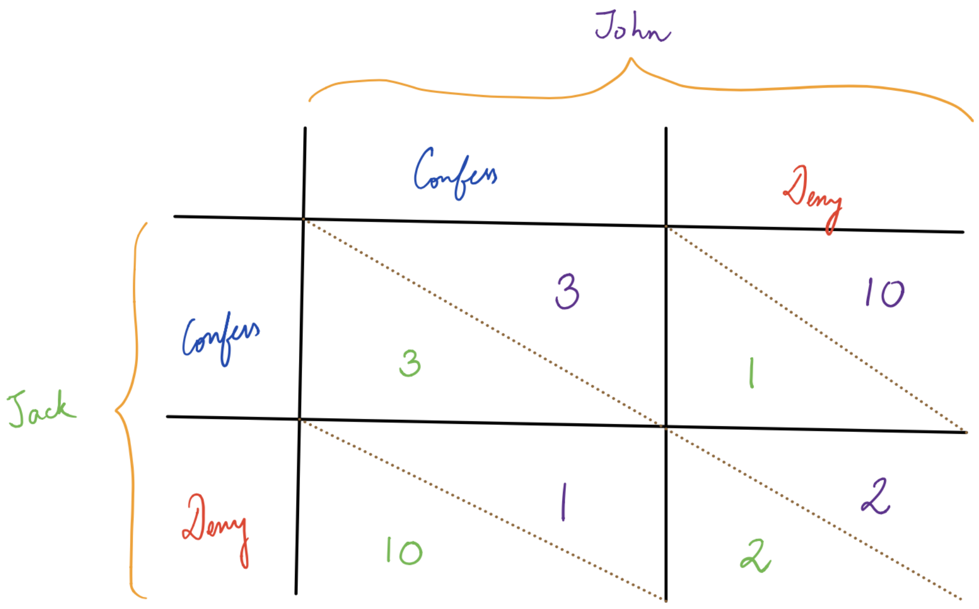
The Prisoner's Dilemma

Suppose that Jack and John are two two individuals charged and convicted with a minor case of money laundering. They both have to serve 2 years of prison each. However, the judge has a suspicion that both of them (they are acquaintances) are involved in some armed burglary (serious felony).

Now, the judge has no evidence present with him at hand. So, he proposes the following to both of them:

- if both of you deny involvement in the burglary, then both of you receive only 2 years of prison each
- if one confesses while the other denies, then the one who confessed gets 1 year while the other gets 10 years of prison
- if both of you confess, then both of you receive 3 years of prison each

Assuming that the Jack and John have no layalty amongst themselves, we should observe that they will pick a non-optimal scenario.



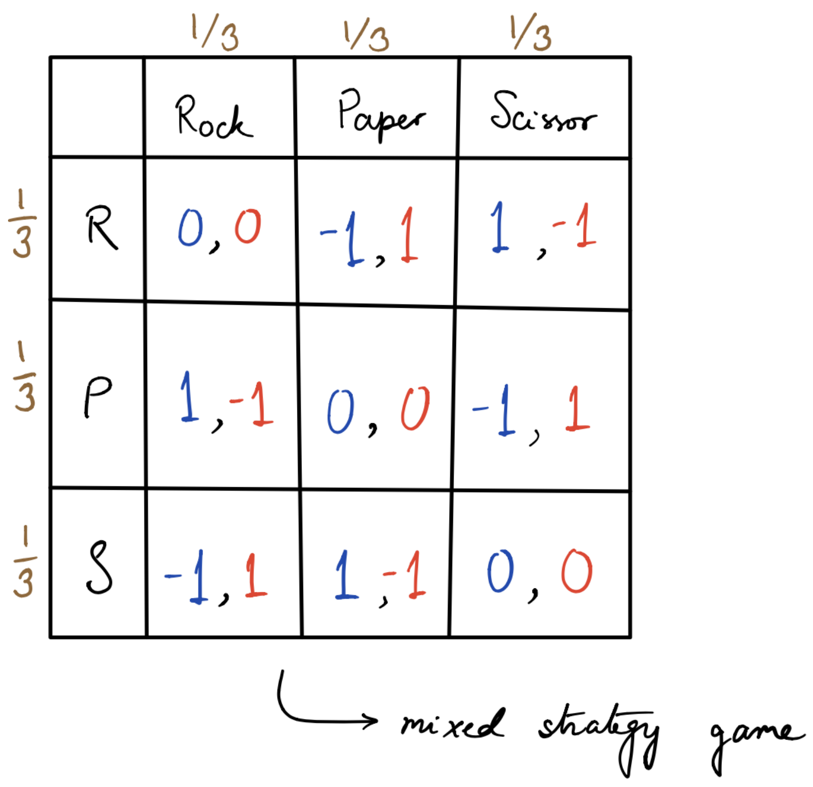
Here, there exists a global optimum in the case where both the prisoners lie. However, given our predisposed knowledge of the situation it might not be the best rational choice for the prisoners.

The best rational choice for the prisoners would be a sub-optimal choice presented in the case where both of them confess. The case serves as a **sub-optimal stable equilibrium** and is referred to as the Nash Equilibrium.

Rock-Paper-Scissors

Let us try the above payoff matrix method shown above for the popular game "Rock-Paper-Scissors".

If you have played the game for a lot of times, you may already have a good intuition as to which scenario will give rise to a Nash Equilibrium.



		1/3	1/3	1/3
		Rock	Paper	Scissor
$\frac{1}{3}$	R	0, 0	-1, 1	1, -1
$\frac{1}{3}$	P	1, -1	0, 0	-1, 1
$\frac{1}{3}$	S	-1, 1	1, -1	0, 0

mixed strategy game

Given the payoff matrix, it is evident that only when both players randomize with equal probability, we obtain a Nash Equilibrium.

Interestingly, this game also serves as a proof by contradiction for the statement that not all games have a Pure Nash Equilibrium. So what's a Pure Nash Equilibrium? Let's find out.

Games: Definitions and Notations

Games can be defined with a set of **players**, where each player has his own set of allowed **actions** (known as **pure strategies**). Any combination of actions will result in a numerical **payoff** (specified by the game) for each player.

Let the number of players be $1, 2, 3, ..., k$ and S_i denote player i's set of actions. n denotes the size of the largest S_i. If k is a constant we look for algorithms that are $O(n^{\kappa})$ where $\kappa \in \mathbb{N}$.

Let us look at the above statements in the light of the game of Rock-Paper-Scissors which was discussed above.

- players, $p = \{1, 2\}$
- $\forall i \in p,\ S_i = \{\text{rock}, \text{paper}, \text{scissor}\}$
- Here, of course $k = 2$, which actually is one of the most studied special case in game theory.
- $n = 3$

Let $S = S_1 \times S_2 \times S_3 \times ... \times S_k$. Then, S is the set of **pure strategy profiles**. Then, each $s \in S$ gives rise to payoff to each player and u_s^i denotes the payoff to player i when all players choose s.

- The list of all possible u_s^i's yields a **normal-form game**, which for a k-player game should be a list of kn^k numbers.
- We have have other models too. For example, a **Bayesian game** is one where u_s^i can be a probability distribution over i's payoff.

Nash Equilibrium

A Nash Equilibrium is a set of strategies - one strategy per player - such that no player has an incentive to unilaterally change his strategy (stable sub-optima). In case of two player zero-sum games, Nash Equilibrium is also optimal.

- **Pure** NE: where each player chooses a pure strategy
- **Mixed** NE: where for each player a probability distribution over his pure strategies is applied (in case of MNE we have expected payoff's associated with each player)

Formal Definition of Nash Equilibrium

From the above, $u_s^i = u_i(s)$ where $s \in S$.

An **individual mixed strategy** is a probability distribution on the set of available strategies. Such as in the last example, selecting one of "rock", "paper", or "scissors" uniformly at random is an example of a mixed strategy.

A **pure strategy** is simply a **special case** of a mixed strategy, in which one strategy is chosen 100% of the time.

- A **Nash equilibrium** is a strategy profile $s = (s_1, s_2, \ldots, s_n)$ with the property that,

$$u_i(s) \geq u_i((s_1, s_2, \ldots, s_i', \ldots, s_n))$$

$\forall i$, where $s_i' \in S_i$ denotes a strategy other than s_i available to player i.

In the event that this inequality is strict, $u_i(s) > u_i((s_1, s_2, \ldots, s_i', \ldots, s_n))\ \forall i$, the profile s is called a **Strong Nash Equilibrium.** Otherwise, s is called a **Weak Nash equilibrium**.

Existence Theorem

Any game with a finite set of players and finite set of strategies has a Nash Equilibrium of Mixed Strategies or MNE.

- Nash's existence theorem guarantees that as long as S_i is finite $\forall i$ and there are a finite number of players (p), at least one Mixed Strategy Nash equilibrium exists.

Some Computational Problems

First Decision Problem

Input: A game in normal form.

Question: Is there a Pure Nash Equilibrium?

Special Case Solutions:

- Determining whether a strategic game has a pure Nash equilibrium is NP-complete given that the game is presented in GNF having a bounded neighborhood or in the case of acyclic-graph or acyclic-hypergraph games.
- Pure Nash Equilibrium existence and computations problems are feasible in logarithmic space for games in standard normal form.

A Second Better Problem

Input: A game in normal form.

Output: Is there a Mixed Nash Equilibrium?

Following the steps of quite a many papers and books written on this topic, we shall call this problem NASH.

Speculation

By Nash's Existence Theorem, this is intrinsically a search problem and not a decision problem.

This might tempt one to look for efficient algorithms for the above problem. There were many attempts in that direction.

> John von Neumann in the 1920s that MNE can
> be formulated in terms of linear programming for zero-sum games.

But what about games that are not zero-sum?

Several algorithms were proposed over the next half century, but all of them are either of unknown complexity, or known to require, in the worst case, exponential time.

Well then, one might ask, is NASH NP-complete?

Complexity of NASH

Equal-Subsets

Input: $A = \{a_1, \ldots, a_n\}$ where $\Sigma_{\forall i} a_i < 2^n - 1$ and $a_i \in \mathbb{Z}$, $\forall i$

Output: Two distinct sets $A_1,\ A_2$ such that $\Sigma_{\forall j \in A_1} j = \Sigma_{\forall k \in A_2} k$.

Before we further discuss NASH let's talk about the problem of EQUAL-SUBSETS.

It is evident that **EQUAL-SUBSETS** $\in$ **NP**, just like NASH. But is it NP-hard too or in other words - **is it reducible from SAT**? Is it NP-complete?

Total Search Problems

EQUAL-SUBSETS and NASH form a very specific group of problems which appear to be different from typical NP-complete problems like SAT.

Why? **Because unlike SAT,** these problems have a **guaranteed solution.** These problems (EQUAL-SUBSETS, NASH etc.) thus are Total Search Problems - problems where every instance has a solution. But is this difference enough to conclude that EQUAL-SUBSETS and NASH are not NP-complete?

Proof: **Total Search Problems in NP are not NP-complete**

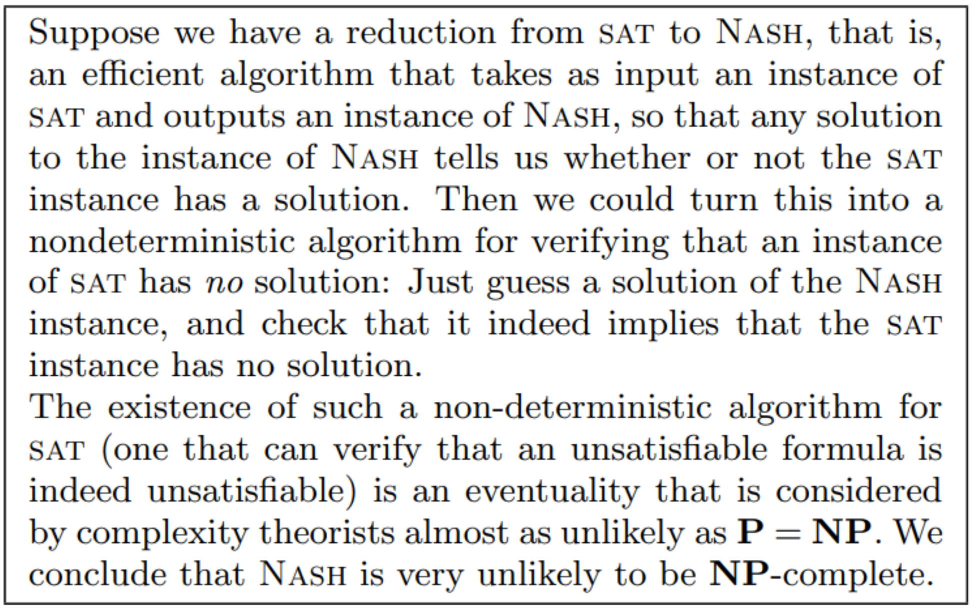
Suppose Ψ is a total search problem in NP which is also NP-Complete. Then, SAT $\preceq_p$ Ψ and $\exists$ an algorithm A for SAT that runs in polynomial time, provided that it has access to poly-time algorithm A' for Ψ.

Now, suppose A has a non-satisfiable formula ϕ which calls A' some number of times. the algorithm eventually returns the answer NO implying that ϕ is not satisfiable.

Now, further suppose that we replace A' with a similar non-deterministic algorithm which in case of EQUAL-SUBSETS would be guess-and-test. This gives us a non-deterministic poly-time algorithm for SAT. The entire algorithm can recognize this non-satisfiable formula ϕ, as before and we have a **NP algorithm that recognizes unsatisfiable formulae.**

This implies $\mathbf{NP} = \mathbf{co\text{-}NP}.$

And, this is specifically why NASH too is very unlikely to be NP-complete. Below, we have another excellent argument by Megiddo.



Suppose we have a reduction from SAT to NASH, that is, an efficient algorithm that takes as input an instance of SAT and outputs an instance of NASH, so that any solution to the instance of NASH tells us whether or not the SAT instance has a solution. Then we could turn this into a nondeterministic algorithm for verifying that an instance of SAT has *no* solution: Just guess a solution of the NASH instance, and check that it indeed implies that the SAT instance has no solution.
The existence of such a non-deterministic algorithm for SAT (one that can verify that an unsatisfiable formula is indeed unsatisfiable) is an eventuality that is considered by complexity theorists almost as unlikely as $\mathbf{P} = \mathbf{NP}$. We conclude that NASH is very unlikely to be $\mathbf{NP}$-complete.

This implies that NASH and cimilar problems are highly likely to be easier than NP-complete problems. What what is the complexity class we can associate them to? A question still remains as to is it easy or is it hard?

TFNP: The Complexity Class

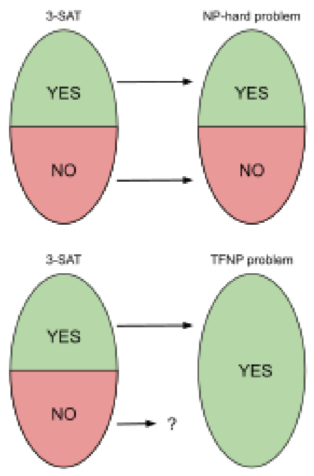
TFNP or Total Function Non-deterministic Polynomial problems are total function problems which can be solved in non-deterministic polynomial time. It is a subset of FNP - a function problem extension of the decision class NP.

This is the complexity class where we shall prove that NASH exists. It also contains the problems FACTORIZATION and EQUAL-SUBSETS.

> TFNP is widely conjectured to contain problems that are computationally intractable.

But, there hasn't been any results yet which show that TFNP problems are NP-hard. Moreover, TFNP-complete problems are not believed to exist.

The top image shows the



typical form of a reduction that shows a problem is NP-hard. Yes instances map to yes instances and no instances map to no instances. The bottom image depicts the intuition for why it is difficult to show TFNP problems are NP-hard. TFNP problems always have a solution and so there is no simple place to map no instances of the original problem. This is the intuition behind the lack of NP-hardness results for TFNP problems.

Proving TFNP Membership

Proving that various search problems are total often use a "non-constructive step" which is hard to compute (unless the problem itself can be efficiently solved). However, it turns out that for most known total search problems these non-constructive steps are one of very few arguments.

These arguments define sub-classes within TFNP since they have mathematical theorems associated that prove their guarantee of existence of solution. TFNP, thus, is often studies through the lenses of these sub-classes and interestingly, these sub-classes have complete problems by virtue of the argument associated evn though TFNP itself does not have known complete problems.

Arguments and Sub-classes

- “If a function maps n elements to n – 1 elements, then there is a collision.” This is the *pigeonhole principle*, and the corresponding class is **PPP**.
- “If a directed graph has an unbalanced node (a vertex with different in-degree and out-degree), then it must have another.” This is the *parity argument* for directed graphs, giving rise to the class **PPAD** considered in this article.
- “If a graph has a node of odd degree, then it must have another.” This is the parity argument, giving rise to the class **PPA**.
- “Every directed acyclic graph must have a sink.” The corresponding class is called **PLS** for *polynomial local search*.

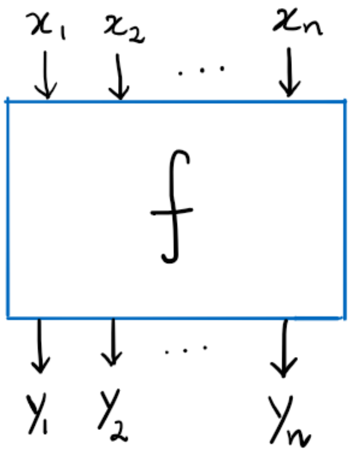
To understand how to define membership to these classes let's go back to our old friend the class NP.

One of the many ways to define NP is *class of all problems that can be reduced into instances of SAT*. Now, we shall define the above subclasses in similar fashion - **define classes** by their **complete problems.**

PPP and EQUAL-SUBSETS

We used the similarity between EQUAL-SUBSETS and NASH to introduce this new class of problems, namely TFNP. Now, it would be quite a shame if we don't actually prove the membership of EQUAL-SUBSETS in TFNP. Moreover, since it is a member of PPP, it will be quite interesting to see how a very simple yet powerful theorem (the pigeonhole principle) that we all have been introduced quite early in high school gives rise to a complexity class of it's own.

PPP stands for *polunomial pigenhole principle*. For the purpose of reductions we construct a gadget in the form of a PIGEONHOLE-CIRCUIT which in itself by definition is a PPP-complete problem.



Input: A boolean circuit C that takes n inputs and n outputs.
Output: A binary vector $\vec{x}$ such that $C(\vec{x}) = 0^n$ or alternatively, 2 vectors $\vec{x}$ and $\vec{x'}$ with $f(\vec{x}) = f(\vec{x'})$.

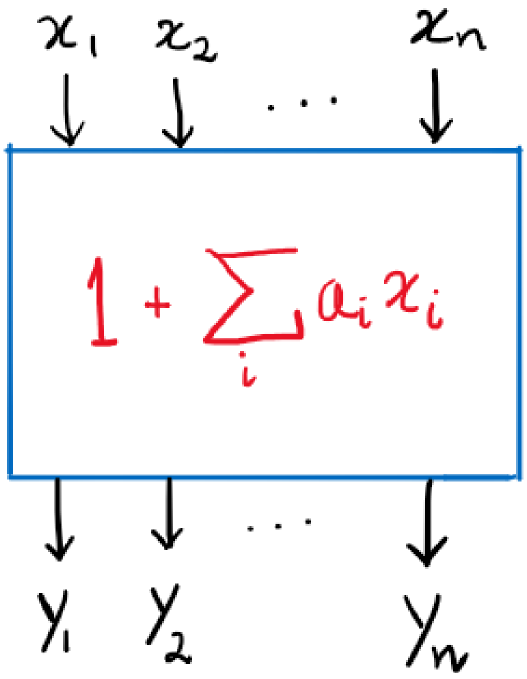
With regard to questions of polynomial time computation, the following are equivalent:

- n inputs/outputs $\rightarrow$ C of size n^2
- n inputs/outputs $\rightarrow$ C of size $O(n^k), k \in \mathbb{N}$
- $n =$ number of gates in C, number of inputs = number of outputs

Definition: A problem Ψ belongs to PPP if Ψ reduces to PIGEONHOLE CIRCUIT in polynomial time. Furthermore, if we can reduce Ψ from PIGEONHOLE CIRCUIT then Ψ is PPP-complete.

EQUAL-SUBSETS belongs to PPP

Taking inspiration from the book Proof without Words, we have:



Example:

Consider, $A = \{2, 3, 4, 5\}$ and $C = 1 + \Sigma_i(a_i x_i) = 1 + \begin{bmatrix} 2 & 3 & 4 & 5 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{bmatrix}$

Then?

We have two vectors 2 vectors $\vec{x} = \begin{bmatrix} 1 \\ 1 \\ 0 \\ 0 \end{bmatrix}$ and $\vec{x'} = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 1 \end{bmatrix}$ such that, $C(\vec{x}) = C(\vec{x'})$.

Why?

Since, $1 + \begin{bmatrix} 2 & 3 & 4 & 5 \end{bmatrix} \begin{bmatrix} 1 \\ 1 \\ 0 \\ 0 \end{bmatrix} = 1 + \begin{bmatrix} 2 & 3 & 4 & 5 \end{bmatrix} \begin{bmatrix} 0 \\ 0 \\ 0 \\ 1 \end{bmatrix} = 5 = (0101)_b$.

Now, before we delve into the next class and classifying NASH let's refine the problem statement bore by NASH.

Refining NASH

Let's state the original problem that we posed.

Input: A game in normal form.

Output: Is there a Mixed Nash Equilibrium?

Now, in this problem for two-player games *the probabilities used by the players are rational numbers* (given that their utilities are also rational). So, it is clear how to write down the solution of a 2-player game.

However, as pointed out in Nash's original paper, when there are more than two players, there may be **only irrational solutions**.

As a result, the problem of computing a Nash equilibrium has to deal with issues concerning numerical accuracy. Therefore, we introduce the concept of Approximate Nash Equilibrium.

Approximate Nash Equilibrium

Input: A game in normal form where, $u_s^p \in [0,1]$.

Output: Is there a Mixed ϵ-Nash Equilibrium?

$$\epsilon\textbf{-Nash Eqm}\text{: } E[\text{payoff}] + \epsilon \geq E[\text{payoff}] \text{ of best possible response}$$

This should be at least as tractable as finding an exact equilibrium, hence any hardness result for approximate equilibria carries over to exact equilibria.

NASH: "Re"-Definition

> Bounded rationality fixes irrationality.

Given the *description of a game* (by explicitly giving the utility of each player corresponding to each strategy profile), and a *rational number* $\epsilon > 0$, compute a *(mixed) Approximate Nash Equilibrium*.

END-OF-A-LINE Problem

Input: A directed graph G and a specified unbalanced vertex of G.

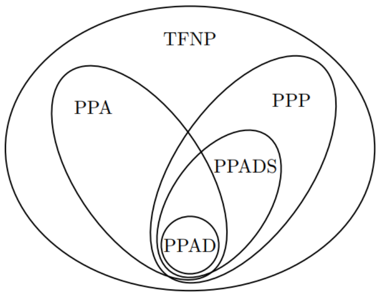
Output: Some other unbalanced vertex.

Now, by parity argument to know that a solution always exists.

> Suppose G has 2^n vertices, one for every bit string of length n (the parameter denoting the size of the problem). For simplicity, we will suppose that every vertex has at most one incoming edge and at most one outgoing edge. The edges of G will be represented by two boolean circuits, of size polynomial in n, each with n input bits and n output bits. Denote the circuits P and S (for predecessor and successor). Our convention is that there is a directed edge from vertex v to vertex v', if given input v, P outputs v' and, vice-versa, given input v', P outputs v. Suppose now that some specific, identified vertex (say, the string $\{0\}^*$) has an outgoing edge but no incoming edge, and is thus unbalanced. With the restriction of at most one incoming and one outgoing edge, the directed graph must be a set of paths and cycles; hence, following the path that starts at the all-zeroes node would eventually lead us to a solution. The catch is, of course, that this may take exponential time. Is there an efficient algorithm for finding another unbalanced node without actually following the path?

PPAD: The Class

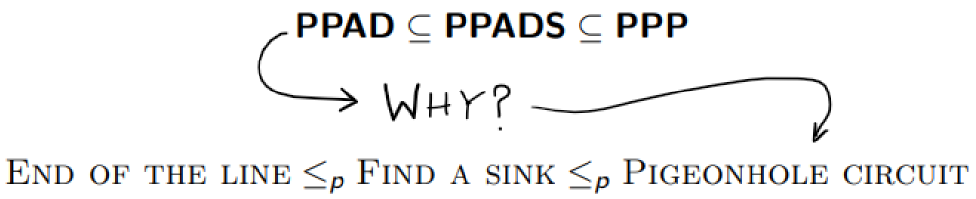
> PPA... what?
>
> **- Papadimitriou**



PPAD (which stands for *Polynomial Parity Arguments on Directed graphs*) is the class of all problems *reducible to END-OF-A-LINE*.

As expected, PPAD problems are believed to be hard, but obtaining PPAD-completeness is a weaker evidence of intractability than that of obtaining NP-completeness.

Arguably, the most celebrated application of the complexity of total search problems is the characterization of the computational complexity of finding a Nash Equilibrium in terms of PPAD-completeness. This problem lies in the heart of game theory and economics. The proof that Nash Equilibrium is PPAD-complete initiated a long line of research in the intersection of computer science and game theory and revealed connections between the two scientific communities that were unknown before.



Does PPAD contain hard problems?

In the absence of a proof P $\neq$ NP, we cannot confirm whether PPAD has hard problems. However, END-OF-A-LINE does appear to be a hard problem. Why?

- END-OF-A-LINE is a hard problem in the query complexity model, i.e., when S, P are given as black-box functions. We are interested in the computational variant where we have access to the circuits. But, except for very few exceptions, we don't know any way of looking inside a circuit and extracting useful information from it.
- Moreover under certain cryptographic assumptions (e.g., indistinguishablility obfuscation) the computational variant of END-OF-LINE is hard.

NASH is PPAD-complete

Here, we shall provide an outline to give an intuition behind why NASH is PPAD-complete.

Two parts of the proof are:

- Nash is in PPAD i.e., $\text{NASH} \preceq_p \text{END-OF-A-LINE}$
- $\text{END-OF-A-LINE} \preceq_p \text{NASH}$ or, NASH is PPAD-hard

Part 1

To prove that NASH is in PPAD, we can use a lot from Nash's original proof of existence, which utilizes Brouwer's fixed point theorem - it is essentially a reduction from NASH to BROUWER (problem of finding an approximate Brouwer fixed point of a continuous function).
Now, BROUWER can be reduced, under certain continuity conditions, to END-OF-A-LINE, and is therefore in PPAD.

$$\text{NASH} \preceq_p \text{BROUWER} \preceq_p \text{END-OF-A-LINE}$$

Let G be a normal form game with k players, $p = \{1, 2, \ldots, k\}$, and strategy sets $S_i = [n]$, $\forall i \in [k]$ and let $\{u_s^i : i \in [k], s \in S\}$ be the utilities of the players.

Also let $\epsilon < 1$. In time polynomial in $|G| + \log(1/\epsilon)$, we will specify two circuits S and P each with $N = \text{poly}(|G|, \log(1/\epsilon))$ input and output bits and $P(0^N) = 0^N \neq S(0^N)$, so that, given any solution to END-OF-A-LINE on input S, P, one can construct in poly-time an ϵ-approximate Nash Equilibrium of G. This is enough for reducing NASH to END-OF-A-LINE with some further observations.

The details of construction of S and P as well as the complete proof has properly been presented in the original 2008 Paper by Daskalakis et. al.

Part 2:

For this part we shall use the result that BROUWER is PPAD-complete and simulate the BROUWER with a game, which effectively reduces BROUWER to NASH.

The PPAD-complete class of Brouwer functions that appear above have the property that their function F can be efficiently computed using arithmetic circuits that are built up from standard operators such as addition, multiplication and comparison. The circuits can be written down as a *data flow graph*, with one of these operators at every node. The key is to simulate each standard operator with a game, and then compose these games according to the *data flow graph* so that the aggregate game simulates the Brouwer function, and an ϵ-Nash equilibrium of the game corresponds to an approximate fixed point of the Brouwer function.

We shall not be adding an elaborate reduction here, which can be found in the original paper as well.

2-NASH is PPAD-complete

Previously, we had discussed why we study Approximate Nash Equilibrium. We had mentioned how for a two-player game, approximation is not required necessarily. Here, we shall concern ourselves with that apparent special case.

Input: A 2-player game in normal form.
Output: Is there a Mixed Nash Equilibrium?

Original version of NASH with 2 players.

Let, this problem be called 2-NASH.

Remember that unlike our redefined NASH, 2-NASH is not mandated to be approximate.

Proof

- 2-NASH to BROUWER: As in before, this is guaranteed by John Nash's original work on the proof of existence of MNE.
- END-OF-A-LINE to BROUWER: This too is evident from the previous discussions.
- BROUWER to IMITATION-GAME: We shall try to reduce IMITATION-GAME from BROUWER. Here, consider the following utility functions.

$$u_1(x,y) = -||x-y||_2^2, \text{ and } u_2(x,y) = -||f(x)-y||_x^2$$

Only when, $y = x = f(y)$ this game has a Nash Equilibrium. However, now both players have infinite actions associated. So, we make the space discrete and both players have exponentially many actions. To fix these, we first reduce to a similar game with $2n$ players with the following utility functions.

$$u_{2i-1}(x_{2i-1}, y_{2i-1}) = -||x_{2i-1} - y_{2i-1}||^2, \text{ and } u_{2i}(x, y_{2i}) = -||f_{2i}(x) - y_{2i}||_x^2$$

- BROUWER to $2n$-NASH: We have $2n$ utility functions as described above and effectively reduce the actions per player to a constant number of actions.
- $2n$-NASH to 2-NASH using LAWYER'S-GAME: Let, Alice and Bob be the lawyers for the players $2i - 1$ and $2i$ respectively. Since, lawyer's choose actions of players we have the following.

$$\text{Alice's actions: } S_1 \times S_3 \times \ldots \times S_{2n-1}$$
$$\text{Bob's actions: } S_2 \times S_4 \times \ldots \times S_{2n}$$

We can furthermore define the utilities of lawyers and make some necessary concessions or arrangements such as forcing the lawyer's to use all of their players. We use HIDE-AND-SEEK to achieve this. Effectively, Alice incentivizes to choose the same index as Bob whereas Bob tries to hide.

$$(u_A)_{MP}(i,j) = \begin{cases} M & \text{if } i = j \\ 0 & \text{otherwise} \end{cases} = -(u_B)_{MP}(i,j)$$

Here, M is a sufficiently large number. Condition of MNE: when Alice and Bob play uniformly at random.

Conclusions

Game Theory encompases a **large genre** of games and auctions. It serves as a formalism for these too. And, we have shown how Nash Equilibrium a fundamental concept in Game Theory is likely to be hard as well as related the field to the study of complexity classes and effectively to computation itself. However, it is **not possible to model** all forms of games especially **simulations and puzzles** (which include 2D platform video games) into Game Theory.

> How about games such as chess?

We can capture this and other similar games (go, checkers) in the present framework by considering two players, Black and White, each with a huge action set containing all possible maps from positions to moves - but of course such formalism

(Game Theory) is **not very helpful for analyzing complexity of chess and similar games**.

Constraint Logic Formalism

Created	@Nov 24, 2020 7:33 AM
Tags	Playing Games for Algo

Introduction

> Its all a game.
>
> **- Erik Demaine**

In Part I, as we prove that Celeste, our original game of choice is NP-complete and in a different case PSPACE-complete by reducing it from **Circuit Formalism** of SAT and TQBF respectively. This is the more or less standard procedure in classic game complexity literature.

The Satisfiability problem (SAT) is by nature *a puzzle*: the player must existentially make a series of variable selections, so that the formula is true. And thus, the corresponding model of computation obtained is non-determinism, and the natural complexity class is **NP**.

When we add existential and universal quantifiers, we create the True Quantified Boolean Formula (TQBF) which has a natural interpretation as a 2-player game. The corresponding model of computation is alternation and the associated class is **PSPACE**.

Furthermore, allowing the players to continue to switch the variable states indefinitely creates a formula game of unbounded length, raising the complexity to **EXPTIME**.

Typically, most hardness results we find, including ours, are direct reductions (many seem more natural than several theoretical problems) from such formula games or their variants.

However, even though these Circuit Formalisms are exremely versatile and reducible into classic puzzles and games, they do suffer a few limitations in certain genres.

Limitations of Circuit Formalism

- Geometric constraints typically found in board games do not naturally correspond to any properties of formula games.
- Extension to higher classes maybe cumbersome.
- Too many gadgets? It can be a problem in certain cases.

Constraint Logic

This new formalism which is originally fomulated by Erik Demaine and Robert Hearn (2009). The main purpose for this formalism is to address the limitations of Circuit Formalism and provide an unifying framework which can be used to prove the hardness of several classes and genres of games - ranging from simulations to puzzles and 2-player games (Board games, Platform video-games, etc.).

Constraint Logic can be considered an intersection of Graph Theory and Circuit Formalism. We have what Demaine refers to as a *constraint graph* and *rules* that define legal moves on such graphs. The games serves as computation when it is played on a constraint graph and simultaneously "serves as a useful problem to reduce to other games to show their hardness".

Advantages

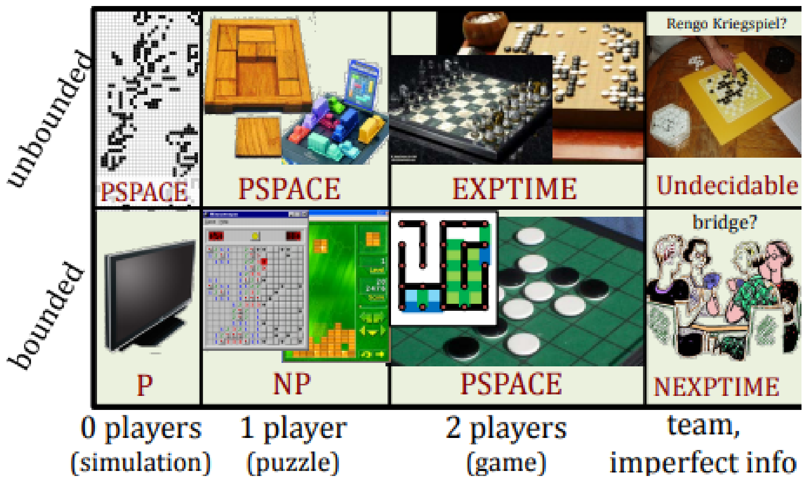
- Planar graphs provide natural correspondence with typical board game topology.
- No variables, formulas or large number of gadgets - only a *constraint graph*. This results in simpler reductions (often).
- Genralizability - more versatile and flexible

> This is awesome.
>
> **- Me**

Now, before we delve into understanding *constraint graphs* and therefore C*onstraint Logic*. Let us look at an amazing generalisation that this theory manages to produce with relative ease (unlike others).

Unbounded	PSPACE	PSPACE	EXPTIME	Undecidable
Bounded	P	NP	PSPACE	NEXPTIME
	Zero player (simulation)	One player (puzzle)	Two player	Team, imperfect information

Okay, now you might be asking yourself questions like what the hell is unboundedness (even though I gave away a slight explanation before). Well, we shall define these terms properly later but to give a taste here goes a better classification. Why better? **Because, it has examples.**



In the next section, we shall introduce the important concepts associated in understanding Constraint Logic formalism. The main target would be to be able to provide a bulk understanding of the topic, in general.

Constraint Logic: Theory

A constraint graph serves as a model of computation, whose orientation describes the machine's state. What does that mean? Well, a machine in this computational model is an undirected graph with red and blue edges. A configuration of the machine gives directions to the edges, thus is a directed graph. Depending on the nature of the game (further constraints), the computation performed can be deterministic, non-deterministic, alternating etc. The constraint graph then accepts the computation just when the game can be won.

Definition: A *constraint graph* G is a directed graph, with edge weights $\in \{1, 2\}$. An edge is called *red* or *blue* respectively. The *inflow* at each vertex is the sum of the weights on inward-directed edges, and each vertex has a non-negative *minimum inflow* ($= 2$).

Legal Configuration: $\text{inflow}_i \geq \text{min-inflow}(= 2),\ \forall i \in V(G)$ which serves as the constraint. We can further assume, $\deg_i \leq 3,\ \forall i \in V(G)$ or in other words, G is 3-regular.

Legal Move: A legal move on a constraint graph is the reversal of a single edge, resulting in a legal configuration.

The Game: Reverse a given edge, by executing a sequence of moves. In multiplayer games, each edge is controlled by an individual player and each player has his own goal edge.

Deterministic Game: An unique sequence of reversals is forced.

Bounded Game: Each edge may only reverse once.

Major Theorems

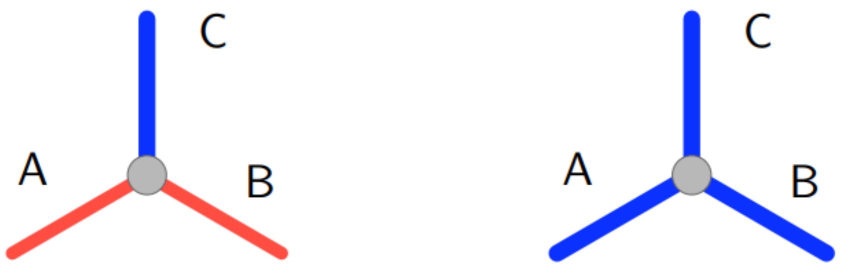
1. **Existence**: The existence of a configuration for a given machine is NP-complete. A modified version would be to prove that Bounded Non-deterministic Constraint Logic is NP-complete.
2. **Reversibility Problem**: Given a constraint graph, can we reverse a specified edge by a sequence of valid moves, is PSPACE-complete.
3. **Reachability Problem**: The reachability problem, that is, given two configurations C_1 and C_2, can we reach C_2 from C_1 by a sequence of valid moves, is PSPACE-complete.

Note: The above two theorems hold for *planar graphs* in which each vertex touches either 3-*blue edges* or 2-*red*-1-*blue edges.*

The Gadgets

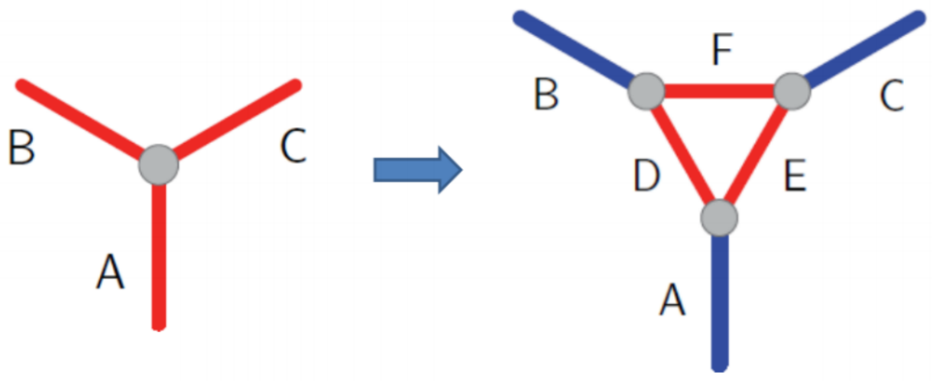
AND vertex: C may be directed outward iff A and B are both directed inward.

OR vertex: C may be directed outward iff either A or B are directed inward.

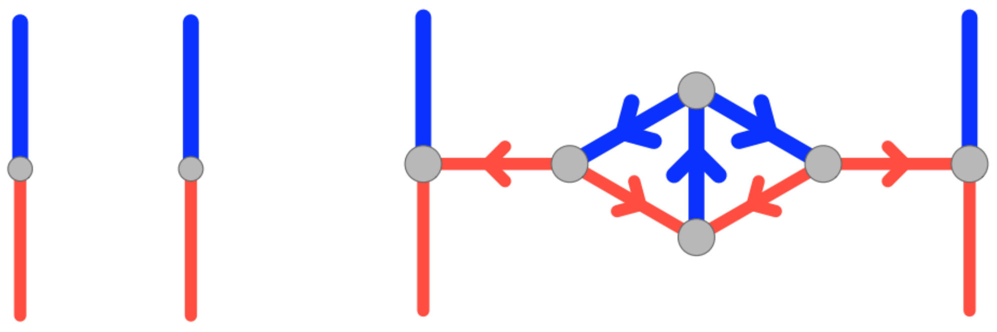


SPLIT vertex gives an alternative view of the AND vertex. It takes 2 red edges as outputs and 1 blue edge as input. The outputs can be active only if the input is active.

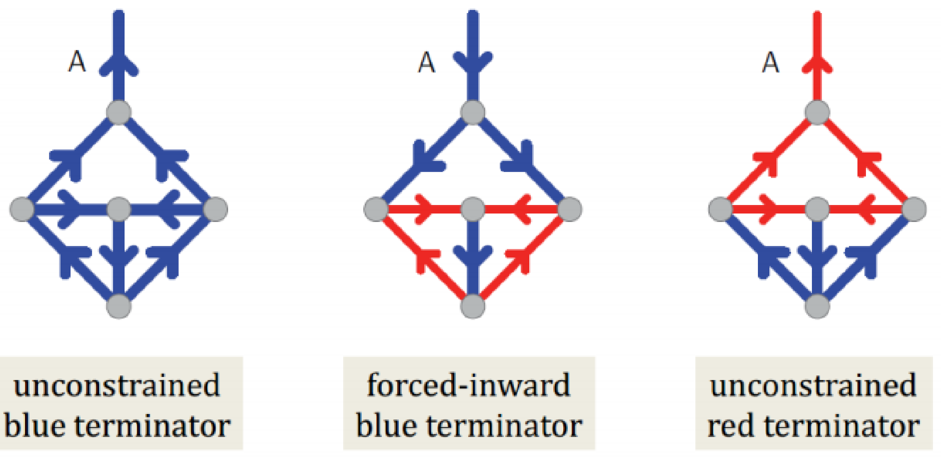
CHOICE vertex: Represented as a node with one red input edge and two red output edges. The idea is that if the input is active, only one of the two outputs can be active, hence the choice. An equivalent construction would use one SPLIT as input and two ANDs as outputs.



RED-BLUE CONVERSION gadget: is used to turn the output of an AND or an OR vertex into
the input for an AND or CHOICE vertex. This changes the color of the edge from blue to
red.



WIRE TERMINATOR gadget: Takes a 1 degree vertex and turns it into a 3 degree vertex. Special cases arise depending on the color and desired orientation of the wire.

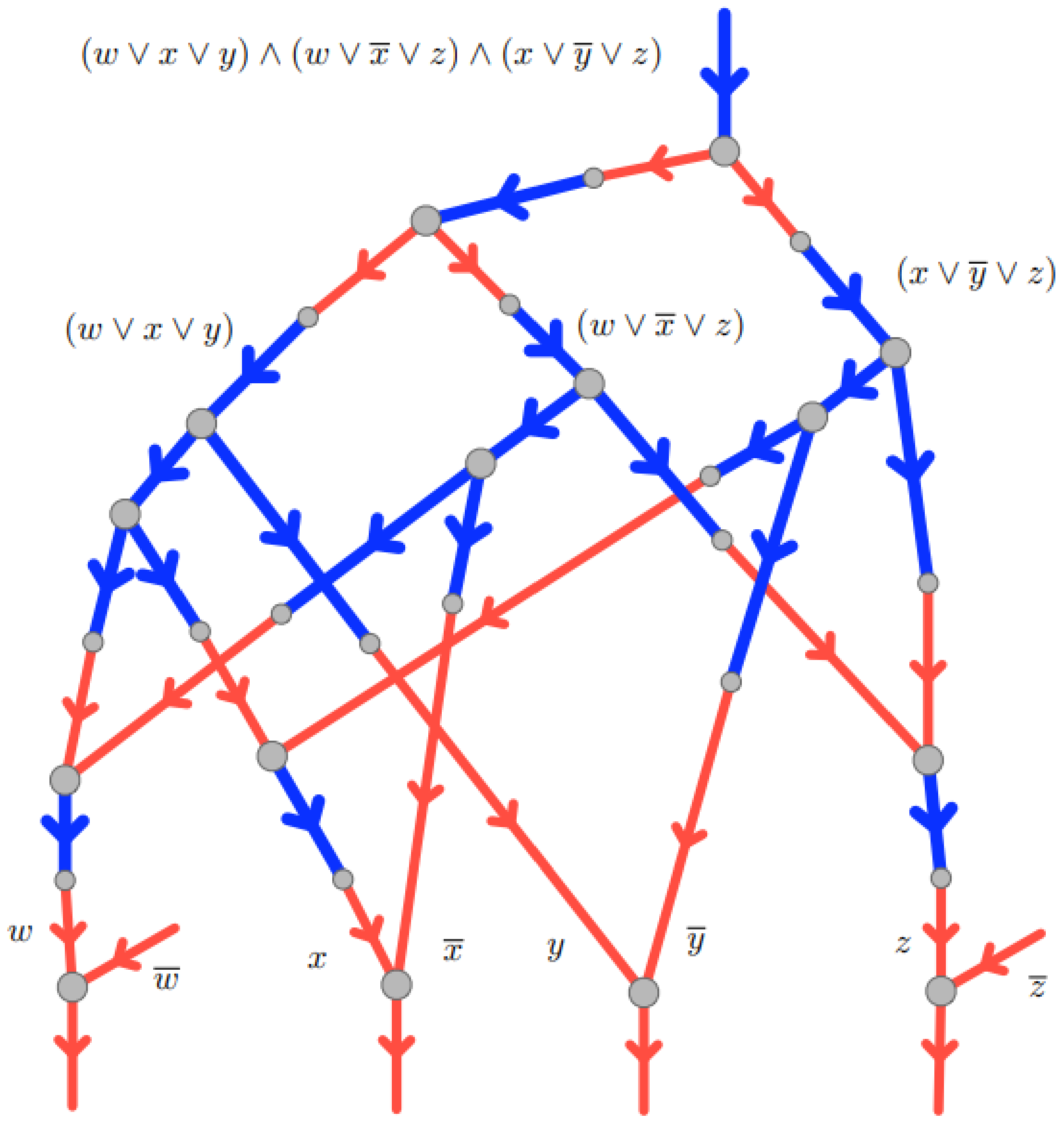

A
A
A
unconstrained blue terminator
forced-inward blue terminator
unconstrained red terminator

Non-Deterministic Constraint Logic

Non-deterministic Constraint Logic serves as the one-player version of Constraint Logic. Since, most of our work on this project deals with Single-player games - especially 2-D Platform video games we shall discuss this first and then move on to simulations and multiplayer games.

The following two proofs serve as the backbone for proving the hardness of puzzles and single player platform video games, which essentially are digital puzzles.

Proof: Bounded NCL is NP-complete


$(w \vee x \vee y) \wedge (w \vee \overline{x} \vee z) \wedge (x \vee \overline{y} \vee z)$
$(x \vee \overline{y} \vee z)$
$(w \vee x \vee y)$
$(w \vee \overline{x} \vee z)$
w
$\overline{w}$
x
$\overline{x}$
y
$\overline{y}$
z
$\overline{z}$

Given an instance of 3-SAT, we construct graph G' as described above. Let F be the corresponding boolean formula. Then, clearly F is satisfiable, the CHOICE vertex edges may be reversed in correspondence with a satisfying assignment, such that the output edge may eventually be reversed. Similarly, if the output edge may be reversed, then a satisfying assignment may be read directly off the CHOICE vertex outputs. Now, $\exists$ an easy poly-time reduction from G' to G where we replace the vertices with 3-*blue-edge* vertex or 2-*red*-1-*blue-edges* vertex. Thus, Bounded NCL is NP-Hard.

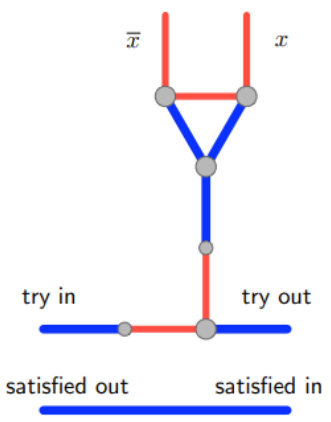
Thus, a proper interpretation of the above proof should be that **Bounded NCL is NP-complete.** Moreover Bounded NCL is NP-complete for planar graphs as well.

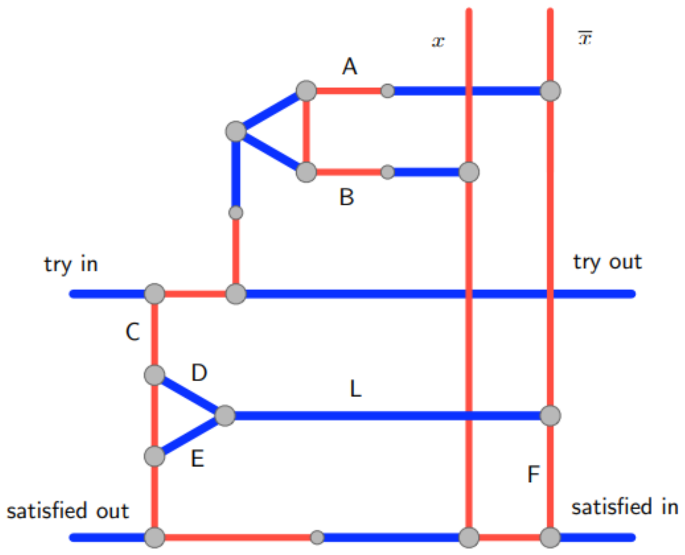
This proof serves as the backbone for proving that Bounded Puzzles are NP-Complete (for example, the first version of our CELESTE proof).

Proof: NCL is PSPACE-complete

NCL is in PSPACE because the state of the constraint graph can be described in a linear number of bits, specifying the direction of each edge, and the list of possible moves from any state can be computed in polynomial time.
Thus, we can have an NPSPACE algorithm where we non-deterministically traverse the state space while maintaining only the current state. Now, by Savitch's Theorem ($\mathrm{PSPACE} = \mathrm{NPSPACE}$) for this NPSPCE algorithm there exists a PSPACE algorithm into which the previous can be converted.

Moreover, NCL is also PSPACE-hard because deciding whether an edge may reverse also decides the TQBF problem. For the complete proof we require gadgets for the quantifiers.





The above are the required EXISTENTIAL and UNIVERSAL quantifier gadgets respectively. Since, we have already had an elaborate reduction from TQBF worked out for CELESTE we leave the rest of the proof to the reader.

Games which are not Puzzles

Here, we shall discuss simulations and 2-player games and then quickly move on to looking at the whole picture (genralization) before concluding.

Simulations

One of the most famous simulations is Conway's Game of Life. This game in our formulism is a Unbounded DCL or Deterministic Constraint Graph. Just as Non-deterministic Constraint Logic serves as the one-player version of Constraint Logic, the zero-player version is DCL (as expected).

Let's talk about the results for simulation:

1. *Bounded DCL is P-complete*: This result is quite boring for the simple fact that there are no interesting games that tend to lie in this genre.
2. *DCL is PSPACE-complete*: This result can be used to model Game of Life and provide a simpler proof as to why it is PSPACE-complete.

2-Player Games

> I have come to the personal conclusion that while all artists are not chess players, all chess players are artists.
> **– Marcel Duchamp**

On that note, let's talk about chess. Suppose, that our realm in only 8×8 chess with all its classic rules, expect one - the 50 move rule, which we shall ignore here.

So, which class does 8×8 classic chess falls in? Well, it does fall in PSPACE as it takes constant space. But, that is not very interesting is it? So, what about $n \times n$ chess or generalised chess?

Bounded and Unbounded 2-Player Games

We have seen how moving from deterministic to non-deterministic computation creates a new and useful model of computation but what if we add an even larger degree of non-determinism (moving from 1-player to 2-player)? Well, we achieve *alternation*.

This results in the complexity of bounded games being raised to PSPACE-complete from NP-completeness and that of unbounded ones from PSPACE-completeness to EXPTIME-complete.

The two-player version of Constraint Logic is called Two-Player Constraint Logic or 2CL (very innovative). To create different moves for the two players, Black and White, we label each constraint graph edge as either Black or White. *This is independent of the red/blue coloration, which is simply a shorthand for*

edge weight. Black (White) is allowed to reverse only. Black (White) edges on his
move. Each player has a target edge he is trying to reverse.

Theorems

1. *Bounded 2CL is PSPACE-complete*.
2. *2CL is EXPTIME-complete*.

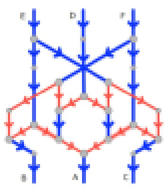
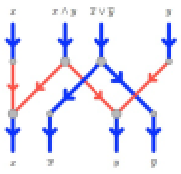
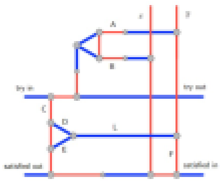
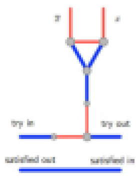
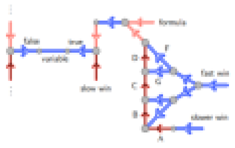
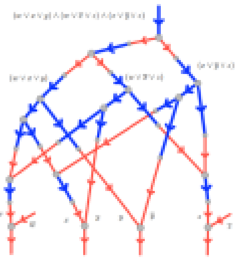
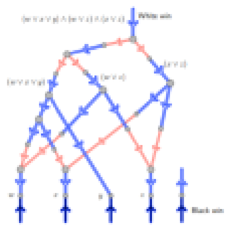
Moreover, just like all previous similar theorems, these hold for Planar Graphs as well.

Interesting Results?

1. Well, it means that generalized versions of chess, go and checkers are provably EXPTIME-complete. A proper reduction from suitable constraint logic graph would do it.
2. For the first time, we have arrived at a class where, $\mathbf{P} \neq \mathbf{EXPTIME}$. Thus, generalised chess is provably intractable, unlike all other games that we have encountered in this project before.

Generalization and Conclusion

Finally, we have got a taste of how beautiful and powerful this new formalism is. Also now, we can further appreciate the several decades of work in Games and Complexity that have resulted in the generalization as follows.

Unbounded	**PSPACE**  	**PSPACE**  	**EXPTIME** 	**Undecidable** **Undecidable**
Bounded	**P** **404**	**NP** 	**PSPACE** 	**NEXPTIME** **NEXPTIME**
	Zero player (Simulation)	**One player (Puzzle)**	**Two player**	**Team, Imperfect information**

An Analysis of Game Design

Discussion on the Design Aspects of Platformers

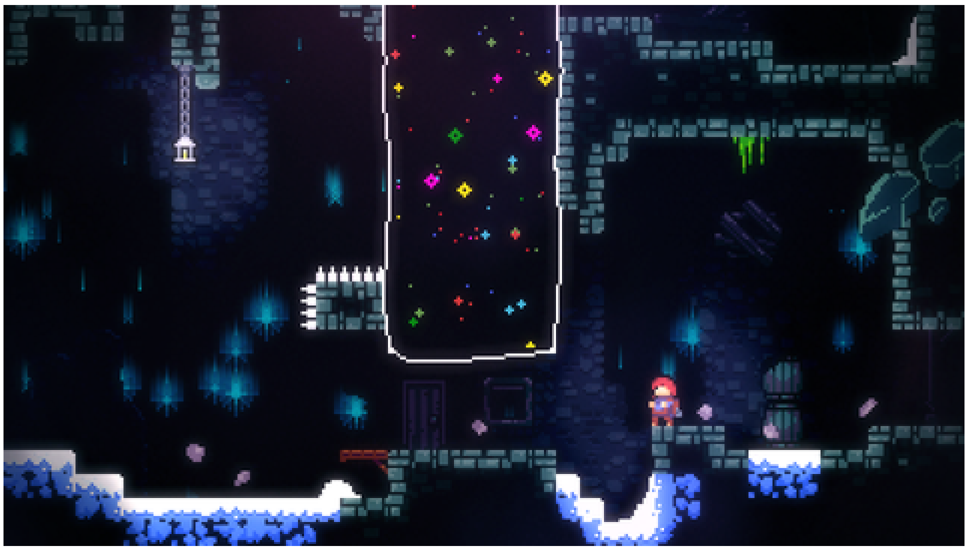
Before talking about how to **generalize hardness results** concerning 2D Platform Games, let's talk about **how games are designed**.

Introduction

In October 1958, Physicist William Higinbotham created `Tennis for two' which is thought to be the first ever video game. The idea was very simple: Two players on either side of the net adjust the angle of their shots and try to hit the ball over the net. Since then, more and more games were designed, some with very unique aspects to them, leading to the current day, where we have a vast ocean of games with tonnes of unique designs.

In this article, we'll be discussing some of the key design aspects of one specific genre, i.e., platformers.

Platformers



Platformers are 2D, side-scrolling video games where the player controls one character. The objective of platformers is to reach set checkpoints in each level, usually indicating the end of that level.

Examples

Classics like Super Mario Bros., Super Contra, Sonic the Hedgehog and Celeste illustrate the idea of platformers very clearly. We will use them as `role-models' throughout our discussion.

Structural Analysis

Frequent Rewards

Platformers are usually designed with multiple levels (like in Super Mario Bros.). The idea behind multiple levels is to keep the player motivated to keep completing more and more levels. This reward system at small intervals is usually the safer and most commonly used method of designing platformers.

Continuity

Straying slightly from the previous structure are games like Celeste, where the game is designed in a continuous manner. The rewards for completing `levels' are placed relatively far apart. This kind of approach is usually backed with a very strong storyline to keep the player motivated instead of a frequent reward system. In Celeste, specifically, each 'scene' of the game itself is very challenging and hence sort of acts like a level in itself. Overcoming the challenge in each scene acts as an alternative to a level-by-level reward system even though no explicit reward is presented to the player. Along with this, Celeste is backed with a beautiful storyline that keeps the players more than just engaged.

Rewarding with a high score

Apart from this, there are also infinite side-scrollers where the objective is to maximize your score. The motivation for the player to keep playing is to beat his/her own score each time or to beat scores of other players.

Game Over

Most games have a very basic storyline to the least. The game usually ends when the story comes to a conclusive end. The player usually completes the bigger objective that was known from the start (in most cases). For example, in Super Mario Bros., Mario wants to rescue Princess Peach. In Celeste, Madeline wants to climb to the top of a mountain. This sense of closure makes the player feel positive about the game and is possibly the safest strategy that game designers use in hopes that the player leaves positive reviews about the game and to make an overall good impression of the game.

As opposed to this, some designers like to make their video game open-ended. This is not a safe strategy at all as it can lead to dissatisfaction among the players. Then why make it open-ended at all? An open-ended game is, in most cases, backed with an absolutely great story/great gameplay. It works only when the game delivers enough to actually make the player think about the open ending. These kind of thoughts on the game are a sign of `greatness of the game' in the video game community that all game designers aspire for. Note that the kind of open-endings we discussed are different from cliffhangers. An open-end is where the player is left uncertain as to what happened next. The player is never intended to know and the end is left open to interpretation. A cliffhanger, on the other hand is used to create a feeling of suspense, where the mystery element is known to be revealed in the future. This is usually done to create anticipation for the next video game in the series.

Analysis of the game's elements

A games element include enemies/obstacles, power-ups and other things that the player can interact with. We'll discuss some platformer-specific elements.

Ground

This may seem like an obvious aspect to most of the games, but it needs to be discussed as it is the most important element in any platformer. As the name suggests, platformers are based on the fact that the player interacts with the platform (ground). The key aspect of a platformer that separates it from other genres is the placement of the ground.
In most games, the ground is only used as a background factor. It doesn't usually play any role other than having something to stand on.
In platformers, however, the ground's placement itself poses a challenge to the player. The ground itself becomes one of the obstacles for the player. In Super Mario Bros., for example, there are gaps in the ground that Mario has to jump over in order to not die. The ground is also placed in other interesting ways to make getting things like power-ups harder. It is a designer's job to focus primarily on the ground's placement, before the other elements.

Obstacles

Enemies or other forms of obstacles are also an integral part of platformers. The difficulty of a level primarily depends on the way these obstacles are placed and also on how the obstacles interact with the player. The choice of these obstacles and their placement should be done extrememly carefully as putting strong ones at the very beginning of the game could lead the player to think that the game is too hard and be demotivated. A weak set of obstacles placed a long way from the beginning could make the player feel the opposite way and would lead to dissatisfaction. The safest strategy is to place them in an increasing order of difficulty. The obstacles can be of various types, some are destroyable like the enemies in Super Mario, some are not like the spikes in Celeste.

The Player

The player is completely controlled by the player and hence it is important for a game designer what freedoms the player needs to be given. In most platformers, the player is given the freedom of moving left-right and jumping. Here, too, the kind of jump the player makes is very important to the player as it determines the player's mastery of control throughout the entire game. Celeste, in particular, does an amazing job in the jump aspect. Different games also give other freedoms like the ability to fire a gun in Super Contra and shoot fireballs in Super Mario. Celeste gives the ability to dash and climb walls. Sonic gives the ability to roll. It is impossible to list down the possible freedoms as it is upto the designer's creativity.

Death

Deciding how the player dies is also extremely important as this can be the difference between an extremely hard game and an easy one. In games like Super Mario, a very beautiful system is implemented. You die immediately once you hit an enemy. But this can be avoided if you have the power of a mushroom. So this makes the game hard but if you play good enough to conserve a mushroom, then you're at low risk. Also in Super Mario, you have a limit to the number of times you can die, making the game very challenging. In Super Contra, a health system is used. In Celeste, you die immediately when you hit an obstacle. No exceptions. But here, you can try a level infinite number of times. All of these different systems are designed very well, giving the player just the right amount of challenge that fits well with the game.

Conclusion

We discussed some key aspects of platformers and why they are designed the way they are. There are a lot of more aspects that can be talked about but I shall end the discussion here as these are some general aspects that apply to most platformers.

Generalizations: 2D Platform Games

Created	@Nov 24, 2020 2:37 PM
Tags	Playing Games for Algo

Introdution

The analysis of 2-D platform games are of great mathematical and computational interest in the recent times. Many of the traditional games and puzzles have undergone rigorous mathematical analysis over the last century and are proved to be conditionally hard (whether $P \neq NP$) while others are provably intractable (as discussed with respect to chess). A similar approach to video games, in particular 2-D platform games, has been fairly recent and the complexity of many video games are yet to be studied and remains largely unexplored.

In essence, 2-D Platformers are puzzles. As a result they are often complete for either NP or PSPACE. It might even be argued that this factor allows such games to be "humanly interesting" to play.

The compleness analyses that we are interested about are often found un-applicable for modern games, espicially non-platform games. This is because most of modern age games use Turing-equivalent scripting languages that allow designers to have *undecidable puzzles* as a part of the gameplay.

NP-Completeness

> In case of NP-complete games, we have levels whose solution demands some degree of *ingenuity*, but such levels are usually solved within a polynomial number of *manipulations*, and the challenge is merely to find them.

PSPACE-Completeness

> In contrast, the additional complexity of a PSPACE-complete game seems to reside in the presence of levels whose solution requires an exponential number of manipulations, and this may be perceived as a nuisance by the player, as it makes for tediously long playing sessions.

On Platformers

We approach the analysis of 2-D platform games by observing that all such games have certain common features. Any 2-D platform game typically involves a sequential set of puzzles (*levels*). At each level, the player must manipulate his/her avatar to the predesignated end point, performing tasks along the way. The avatar is controlled by the player using a limited simple commands (step, jump, crouch etc.). Further, the avatar is affected by some form of gravity, as a consequence of which maximum jump height is limited and avatar could potentially fall from a height. Each level in such games are represented by a 2-D map representing a vertical slice of a virtual world which usually consists of horizontal platforms, hence the name *platform game*.

In addition, many of these games have certain common features, as enlisted below:

- long fall: The height of the longest safe fall (that does not hurt theavatar), which is taken to be larger than the maximum jump height.
- opening doors: The game world may contain a variable number ofdoors and suitable mechanisms to open them.
- closing doors: The game world contains a mechanism to close doors, and a way to force the player to trigger such a mechanism.
- collecting items: The game world contains items that must be collected.
- enemies: The game world contains enemy characters that must bekilled or avoided in order to solve the level.
- time limit: The given level of the game must be completed within thea given time limit.

These features allow for several generalizations or *metatheorems* concerning these platformers which help us understanding and approaching the problem (*decision problem associated*) with greater abstraction.

While certain or combination of features are easy from an algorithmic point, other features, or certain combinations of thereof, are hard to manipulate. Thus having such a feature within the game essentially imply that the game itself is hard to solve, regardless of other details.

For any given game, these meta-theorems can be adjusted according to details of the particular game. In the next section we state the several metatheorems as well as some relevant conjectures.

Metatheorems

1. *A 2-D platform game where the levels are constant and there is* ***no time limit*** *is in* ***P****, even if the collecting items feature is present.*
2. *Any game exhibiting both* ***location traversal*** *(with or without a starting location or an exit location) and* ***single-use paths*** *is* ***NP-hard****.*
3. *Any 2-D platform game that exhibits the features l****ong fall and opening doors*** *is* ***NP-hard****.*
4. *Any 2-D platform game that exhibits the features* ***long fall, opening doors and closing doors*** *is* ***PSPACE-hard****.*
5. *A game is* ***NP-hard*** *if either of the following holds:*

   (a) The game features ***collectible*** *tokens, toll roads, and location traversal.*
   (b) The game features ***cumulative*** *tokens, toll roads, and location traversal.*
   (c) The game features ***collectible cumulative*** *tokens, toll roads, and the avatar has to reach an exit location.*
6. *A game is* ***NP-hard*** *if it contains* ***doors and one-way paths****, and either of the following holds:*

   (a) The game features ***collectible*** *keys and location traversal.*
   (b) The game features ***cumulative*** *keys and location traversal.*
   (c) The game features ***collectible cumulative*** *keys and the avatar has to reach an exit location.*
7. *If a game features* ***doors and pressure plates****, and the avatar has to reach an exit location in order to win, then:*

   (a) Even if ***no door*** *can be* ***closed by a pressure plate****, and if crossovers are*

*allowed, then the game is **P-hard**.*

*(b) Even if **no two pressure plates** control the same door, the game is **NP-hard**.*

*(c) If **each** door may be controlled by **two pressure plates**, then the game is **PSPACE-hard**.*

8. *If a game features **doors and k-buttons**, and the avatar has to reach an exit location in order to win, then:*

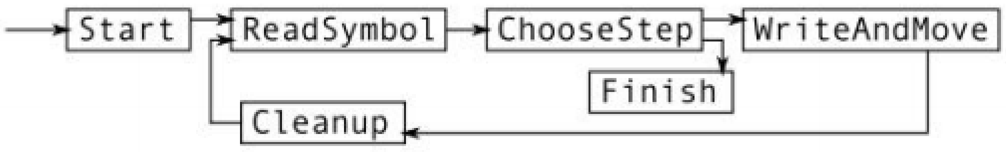
   *(a) If **k > 1,** and crossovers are allowed, then the game is **P-hard.***
   *(b) If **k > 2,** then the game is **NP-hard.***
   *(c) If **k > 3,** then the game is **PSPACE-hard.***

9. *There exists an NP-complete game featuring doors and 2-buttons in which the avatar has to reach an exit location. (One that we end up proving in CELESTE).*

Applications

These results can be use to directly analyse the complexity of several 2-D platform games. For instance, Commander Keen, Crystal Caves, Secret Agent and Bio-Menace can be shown to be **NP-hard** due to presence of switches for activation of moving platforms. Similarly, Jill of the Jungle and Hocus Pocus are also **NP-hard** due to the presence of switches that toggle walls, so are Crash Bandicoot and Jazz Jackrabbit 2, due to crates on breaking which sets of new floor tiles are activated. Duke Nukem with the traditionally observed 4 key-cards will be **P**, while the generalized case with n-keys will be **NP-hard**.

The game *Prince of Persia* is quite interesting. An instance of the word problem for linear bound automata (WordLBA) is reduced to a instance of the game. The general layout of an instance of the game Prince of Persia is as shown below:



The reduction allows us to conclude that the game *Prince of Persia* is **PSPACE-hard**. Further, it is known that the game is in **NPSPACE**. These results, along with Savitch's theorem which states that **NPSPACE = PSPACE**, can be used to show that the game Prince of Persia belongs to **PSPACE-complete**.

The case of the generalised version of the game *Lemmings* is also quite interesting. A proof was given by Cormode in his 2004 paper in which *Lemmings* is shown to belong to **NP**. However, the author of the original article on this topic argues against the proof by noting that the setup used Cormode's proof with an exhaustive trial and error algorithm fails if the number of theoretically possible configurations is not polynomial in the input size. Using the earlier given meta-theorems, the author goes on to show the sequence of instances $\{I_n\}_{n=1}^{\infty}$ is valid for the game *Lemmings* and a counter example to the Cormode's proof. Further, this result is shown to be true for a simple (non generalised version) of the game *Lemming* too. The author goes on to conclude that while Concorde's proof of the game being **NP-hard** is valid, we cannot comment whether the problem belongs **NP** or not.

Conclusion

> Gaming is a hard job, but someone as to do it.
>
> **- Giovanni Vigilietta**

Thus, with the beautiful quote from the author of the original paper that motivates this section, we conclude Part II of our Project. Hopefully, through this endeavour of ours, we have managed to put out the beauty of Complexity Theory through games and their reduction proofs, as well as, to show that understanding games often result in marvellous breakthroughs in not only Computer Science but several other fields as well. Artificial Intelligence serves as a breathing example for this.