NP and coNP

Theorem

Let $L \in NP$ then there exists a polynomial decidable and balanced relation R_L such that for all string $x \in L$: there exists a string y such that $R_L(x, y)$ if and only if $x \in L$.

NP and coNP

Theorem

Let $L \in NP$ then there exists a polynomial decidable and balanced relation R_L such that for all string $x \in L$: there exists a string y such that $R_L(x, y)$ if and only if $x \in L$.

- $L \in NP$ iff $\overline{L} \in coNP$
- if $L \in NP$ then all $x \in L$ have succint certificates.
- if $\overline{L} \in coNP$ then all $x \in \overline{L}$ have succint disqualifications.

NP and coNP

Theorem

Let $L \in NP$ then there exists a polynomial decidable and balanced relation R_L such that for all string $x \in L$: there exists a string y such that $R_L(x, y)$ if and only if $x \in L$.

- $L \in NP$ iff $\overline{L} \in coNP$
- if $L \in NP$ then all $x \in L$ have succint certificates.
- if $\overline{L} \in coNP$ then all $x \in \overline{L}$ have succint disqualifications.

A "no"-instance of a problem in coNP possesses a short proof of being a "no"-instance.

NP and coNP

Validity $\in$ coNP: all Boolean expressions ϕ that are valid

If ϕ is not valid formula then there is an assignment such that $\phi = \text{FALSE}$.

NP and coNP

Validity $\in$ coNP: all Boolean expressions ϕ that are valid

If ϕ is not valid formula then there is an assignment such that $\phi =$ FALSE.

HAMILTON PATH COMPLEMENT $\in$ coNP: all graphs that have no Hamilton path.

All "no"-instances of HAMILTON PATH COMPLEMENT have at least one Hamilton path.

NP and coNP

Validity $\in$ coNP: all Boolean expressions ϕ that are valid

If ϕ is not valid formula then there is an assignment such that $\phi =$ FALSE.

HAMILTON PATH COMPLEMENT $\in$ coNP: all graphs that have no Hamilton path.

All "no"-instances of HAMILTON PATH COMPLEMENT have at least one Hamilton path.

Theorem

Validity and HAMILTON PATH COMPLEMENT are coNP-complete problems.

NP and coNP

Theorem

If L is NP-complete problem then $\overline{L} = \Sigma^* - L$ *is coNP-complete problem.*

NP and coNP

Theorem

If L is NP-complete problem then $\overline{L} = \Sigma^* - L$ *is coNP-complete problem.*

Theorem

$P \subseteq NP \cap coNP$

Proof.

$P \subseteq NP$ and P is closed are complement. Thus, $P \subseteq coNP$. As a result, $P \subseteq NP \cap coNP$. □

NP and coNP

Theorem

If L is NP-complete problem then $\overline{L} = \Sigma^* - L$ *is coNP-complete problem.*

Theorem

$P \subseteq NP \cap coNP$

Proof.

$P \subseteq NP$ and P is closed are complement. Thus, $P \subseteq coNP$. As a result, $P \subseteq NP \cap coNP$. □

We don't know whether $P = NP \cap coNP$.

$NP \cap coNP$

Let $L \in NP \cap coNP$: Each "yes"-instance of L has a succint certificate and each "no"-instance of L has a succint disqualification.

$NP \cap coNP$

Let $L \in NP \cap coNP$: Each "yes"-instance of L has a succint certificate and each "no"-instance of L has a succint disqualification.

Theorem

$PRIMES \in NP \cap coNP$

$NP \cap coNP$

Let $L \in NP \cap coNP$: Each "yes"-instance of L has a succint certificate and each "no"-instance of L has a succint disqualification.

Theorem

$PRIMES \in NP \cap coNP$

MaxFlow

Given a network G and a goal K, whether there is a flow from s to t of value K

$NP \cap coNP$

Let $L \in NP \cap coNP$: Each "yes"-instance of L has a succint certificate and each "no"-instance of L has a succint disqualification.

Theorem

$PRIMES \in NP \cap coNP$

MaxFlow

Given a network G and a goal K, whether there is a flow from s to t of value K

MinCut

Given a network G and a budget B, whether there is a set of edges of total capacity B or less s.t. deleting these edges disconnects s form t

$NP \cap coNP$

Corollary

$MaxFlow \in NP \cap coNP$

$NP \cap coNP$

Corollary

$MaxFlow \in NP \cap coNP$

Proof.

- Given a *yes*-instance the verifier will be the flow of value K
- Given a *no*-instance the disqualifier will be the cut of capacity $K - 1$

□

$NP \cap coNP$

Corollary

$MaxFlow \in NP \cap coNP$

Proof.

- Given a *yes*-instance the verifier will be the flow of value K
- Given a *no*-instance the disqualifier will be the cut of capacity $K - 1$

□

Unfortunately, both *PRIMES* and *MaxFlow* proved to be in P

Function Problems

We have dealt with decision (yes/no)-problems. We are not only interested in deciding whether a graph has a Hamilton path, but also in finding one.

Function Problems

We have dealt with decision (yes/no)-problems. We are not only interested in deciding whether a graph has a Hamilton path, but also in finding one.

We call such problems Function Problems.

Function Problems

We have dealt with decision (yes/no)-problems. We are not only interested in deciding whether a graph has a Hamilton path, but also in finding one.

We call such problems Function Problems.

Definition

FSAT: Given a Boolean expression ϕ, if ϕ is satisfiable then return a satisfying true assignment of ϕ, else return "no".

Function Problems

We have dealt with decision (yes/no)-problems. We are not only interested in deciding whether a graph has a Hamilton path, but also in finding one.

We call such problems Function Problems.

Definition

FSAT: Given a Boolean expression ϕ, if ϕ is satisfiable then return a satisfying true assignment of ϕ, else return "no".

FSAT is as hard as SAT (Why?)

Function Problems

Let $L \in NP$ then there exists a polynomial decidable and balanced relation R_L such that for all string $x \in L$: there exists a string y such that $R_L(x, y)$ if and only if $x \in L$.

Function Problems

Let $L \in NP$ then there exists a polynomial decidable and balanced relation R_L such that for all string $x \in L$: there exists a string y such that $R_L(x, y)$ if and only if $x \in L$.

Let FL be the function problem associated with $L \in NP$

Definition

FL: Given x, find y such that $R_L(x, y)$ if such string exists; else return "no".

Function Problems

Let $L \in NP$ then there exists a polynomial decidable and balanced relation R_L such that for all string $x \in L$: there exists a string y such that $R_L(x, y)$ if and only if $x \in L$.

Let FL be the function problem associated with $L \in NP$

Definition

FL: Given x, find y such that $R_L(x, y)$ if such string exists; else return "no".

Definition

The class of functions associated as above with languages in NP is called FNP

FP and Reductions

Definition

FP $\subseteq$ FNP is the subclass of problems in FNP that can be solved in polynomial time.

FP and Reductions

Definition

FP $\subseteq$ FNP is the subclass of problems in FNP that can be solved in polynomial time.

Definition

A functional problem A reduces to functional problem B if:
There exist string function R,S s.t.:

- *R,S computable functions in logarithmic space*
- *for any x instance of A: the string $z = R(x)$ is an instance of B*
- *S(z) is the correct output of the input x*

FP and Reductions

Definition

FP ⊆ FNP is the subclass of problems in FNP that can be solved in polynomial time.

Definition

A functional problem A reduces to functional problem B if:
There exist string function R,S s.t.:

- *R,S computable functions in logarithmic space*
- *for any x instance of A: the string z = R(x) is an instance of B*
- *S(z) is the correct output of the input x*

TFNP

Theorem

FSAT is FNP-complete.

TFNP

Theorem

FSAT is FNP-complete.

Theorem

FP= FNP if and only if P=NP.

TFNP

Theorem

FSAT is FNP-complete.

Theorem

FP= FNP if and only if P=NP.

Definition

A "Total" FPN problem(TFNP) is a problem whose solution always exists.

TFNP

Theorem

FSAT is FNP-complete.

Theorem

FP= FNP if and only if P=NP.

Definition

A "Total" FPN problem(TFNP) is a problem whose solution always exists.

TFNP

A problem FL is in TFNP if for every string x there exists at least one string y such that $R_L(x, y)$.

TFNP

Primality $\in$ TFNP

Primality $\in$ TFNP: Given a integer N find its prime decomposition $N = p_1^{k_1} \cdot p_2^{k_2} ... p_n^{k_n}$ together with the primality certificates $p_1, p_2, ..., p_n$.

TFNP

Primality $\in$ TFNP

Primality $\in$ TFNP: Given a integer N find its prime decomposition $N = p_1^{k_1} \cdot p_2^{k_2} ... p_n^{k_n}$ together with the primality certificates $p_1, p_2, ..., p_n$.

It is certain that every integer N has a prime decomposition, but it is not believed that there exists polynomial time algorithm.

TFNP

Primality $\in$ TFNP

Primality $\in$ TFNP: Given a integer N find its prime decomposition $N = p_1^{k_1} \cdot p_2^{k_2} ... p_n^{k_n}$ together with the primality certificates $p_1, p_2, ..., p_n$.

It is certain that every integer N has a prime decomposition, but it is not believed that there exists polynomial time algorithm.

Theorem

$FP \subseteq TFNP \subseteq FNP$

TFNP

Primality $\in$ TFNP

Primality $\in$ TFNP: Given a integer N find its prime decomposition $N = p_1^{k_1} \cdot p_2^{k_2} ... p_n^{k_n}$ together with the primality certificates $p_1, p_2, ..., p_n$.

It is certain that every integer N has a prime decomposition, but it is not believed that there exists polynomial time algorithm.

Theorem

$FP \subseteq TFNP \subseteq FNP$

- $\mathrm{FP} = \mathrm{TFNP} \Longrightarrow \mathrm{P} = \mathrm{NP} \cap \mathrm{coNP}$
- $\mathrm{TFNP} = \mathrm{FNP} \Longrightarrow \mathrm{NP} = \mathrm{coNP}$

TFNP = FP?

- *We are looking for problems in TFNP - FP*
- *TFNP-complete problems are candidates*
- *It is believed that there are no TFNP-complete problems!!*

TFNP $=$ FP?

- *We are looking for problems in TFNP - FP*
- *TFNP-complete problems are candidates*
- *It is believed that there are no TFNP-complete problems!!*

What we will do?

TFNP = FP?

- *We are looking for problems in TFNP - FP*
- *TFNP-complete problems are candidates*
- *It is believed that there are no TFNP-complete problems!!*

What we will do?

We can define subclasses of TFNP that have complete problems. The complete problems for these classes are good candidates.

TFNP = FP?

- *We are looking for problems in TFNP - FP*
- *TFNP-complete problems are candidates*
- *It is believed that there are no TFNP-complete problems!!*

What we will do?

We can define subclasses of TFNP that have complete problems. The complete problems for these classes are good candidates.

We will present two of these classes:

- PLS
- PPAD

Polynomial Time Local Search (PLS)

Generally

Polynomial Local Search (PLS) is a complexity class that models the difficulty of finding a locally optimal solution to an optimization problem.

Polynomial Time Local Search (PLS)

Generally

Polynomial Local Search (PLS) is a complexity class that models the difficulty of finding a locally optimal solution to an optimization problem.

In a combinatorial optimization problem A, every instance I has an associated finite set S(I) of solutions, every solution s $\in$ S(I) has a cost $p_I(s)$ that is to be maximized or minimized ,every solution s has a neighbourhood $N_I(s) \subseteq S(I)$.

Polynomial Time Local Search (PLS)

Generally

Polynomial Local Search (PLS) is a complexity class that models the difficulty of finding a locally optimal solution to an optimization problem.

In a combinatorial optimization problem A, every instance I has an associated finite set S(I) of solutions, every solution s $\in$ S(I) has a cost $p_I(s)$ that is to be maximized or minimized ,every solution s has a neighbourhood $N_I(s) \subseteq S(I)$.

Definition

A problem A is in PLS if solutions are polynomially bounded in the input size and there are polynomial-time algorithms for the following tasks:

Polynomial Time Local Search (PLS)

Generally

Polynomial Local Search (PLS) is a complexity class that models the difficulty of finding a locally optimal solution to an optimization problem.

In a combinatorial optimization problem A, every instance I has an associated finite set S(I) of solutions, every solution s $\in$ S(I) has a cost $p_I(s)$ that is to be maximized or minimized ,every solution s has a neighbourhood $N_I(s) \subseteq S(I)$.

Definition

A problem A is in PLS if solutions are polynomially bounded in the input size and there are polynomial-time algorithms for the following tasks:

- *test whether a given string I is an instance of A and if so compute a (initial) solution in S(I)*

Polynomial Time Local Search (PLS)

Generally

Polynomial Local Search (PLS) is a complexity class that models the difficulty of finding a locally optimal solution to an optimization problem.

In a combinatorial optimization problem A, every instance I has an associated finite set S(I) of solutions, every solution s $\in$ S(I) has a cost $p_I(s)$ that is to be maximized or minimized ,every solution s has a neighbourhood $N_I(s) \subseteq S(I)$.

Definition

A problem A is in PLS if solutions are polynomially bounded in the input size and there are polynomial-time algorithms for the following tasks:

- *test whether a given string I is an instance of A and if so compute a (initial) solution in S(I)*
- *given I,s test whether s $\in$ S(I) and if so compute its value $p_I(s)$*

Polynomial Time Local Search (PLS)

Generally

Polynomial Local Search (PLS) is a complexity class that models the difficulty of finding a locally optimal solution to an optimization problem.

In a combinatorial optimization problem A, every instance I has an associated finite set S(I) of solutions, every solution s $\in$ S(I) has a cost $p_I(s)$ that is to be maximized or minimized ,every solution s has a neighbourhood $N_I(s) \subseteq S(I)$.

Definition

A problem A is in PLS if solutions are polynomially bounded in the input size and there are polynomial-time algorithms for the following tasks:

- *test whether a given string I is an instance of A and if so compute a (initial) solution in S(I)*
- *given I,s test whether s $\in$ S(I) and if so compute its value $p_I(s)$*
- *given I,s test whether s is a local optimum and if not, compute a better neighbor $s' \in N_I(s)$.*

Polynomial Time Local Search (PLS)

$PLS \subseteq TFNP$

Polynomial Time Local Search (PLS)

$PLS \subseteq TFNP$

Argument of existence

Every finite directed acyclic graph has a sink.

Polynomial Time Local Search (PLS)

$PLS \subseteq TFNP$

Argument of existence

Every finite directed acyclic graph has a sink.

PLS-complete problems

- Stable configuration for neural networks.
- TSP, under the Kernighan-Lin neighborhood
- MAX-CUT, under the flip neighborhood.

Polynomial Time Local Search (PLS)

$PLS \subseteq TFNP$

Argument of existence

Every finite directed acyclic graph has a sink.

PLS-complete problems

- Stable configuration for neural networks.
- TSP, under the Kernighan-Lin neighborhood
- MAX-CUT, under the flip neighborhood.

These problems are complete in the sense that if there exists a polynomial time algorithm for finding a local optimum for these problems. Then, we can find in polynomial time a local optimum in any problem in PLS.

Stable configuration for neural networks

Example

Stable configuration for neural networks

- $G = (V, E)$
- $S : V \to \{-1, 1\}$ (Nodes)
- Stable Configuration: $\forall i \in V : \quad S(i) \cdot \sum_{\{i,j\} \in E} S(j) \cdot w_{ij} \geq 0$

Define:

- Cost: $c(x, S) = \sum_{\{i,j\} \in E} S(i)S(j)w_{ij}$
- Neigborhood: S' $\in$ N(x,S) $\iff$ Hamming distance(S,S')=1

Stable configuration for neural networks

- *SCNN* $\in$ *TFNP(why?)*
- *SCNN is PLS-complete*
- *the standard algorithm can need exponential number of steps*
- *we don't know a polynomial algorithm for this problem*

Stable configuration for neural networks

- *SCNN* $\in$ *TFNP(why?)*
- *SCNN is PLS-complete*
- *the standard algorithm can need exponential number of steps*
- *we don't know a polynomial algorithm for this problem*

PLS-complete problems

- CIRCUIT FLIP
- STABLE CONFIGURATION FOR A NEURAL NETWORK
- PURE NASH EQUILIBRIUM IN CONGESTION GAMES

Polynomial Parity Arguments on Directed graphs(PPAD)

Let P, N *two boolean circuits that take as an input a* $\{0,1\}^n$ *string and output another* $\{0,1\}^n$*. These two circuits define implicitly a directed graph, where the nodes are the* $\{0,1\}^n$*. There is a directed edge* (v_1, v_2) *iff*

- $P(v_2) = v_1$
- $N(v_1) = v_2$

END OF LINE

Given two circuits P and N as above, if 0^n is an unbalanced node in the graph, find another unbalanced node; otherwise, return "*yes*"

Polynomial Parity Arguments on Directed graphs(PPAD)

Let P, N two boolean circuits that take as an input a $\{0,1\}^n$ string and output another $\{0,1\}^n$. These two circuits define implicitly a directed graph, where the nodes are the $\{0,1\}^n$. There is a directed edge (v_1, v_2) iff

- $P(v_2) = v_1$
- $N(v_1) = v_2$

END OF LINE

Given two circuits P and N as above, if 0^n is an unbalanced node in the graph, find another unbalanced node; otherwise, return "*yes*"

PPAD

Is the class all of search problems that are polynomial-time reducible to END OF LINE

Polynomial Parity Arguments on Directed graphs(PPAD)

$PPAD \subseteq TFNP$

Polynomial Parity Arguments on Directed graphs(PPAD)

$PPAD \subseteq TFNP$

In a directed graph at which $\forall v \in V$:
$1 \leqslant indegree(v) + outdegree(v) \leqslant 2$. Since y_0 has indegree $=0$ and outdegree $=1$. Then, there exists another node y' s.t. $indegree(y') + outdegree(y') = 1$

Polynomial Parity Arguments on Directed graphs(PPAD)

PPAD $\subseteq$ *TFNP*

In a directed graph at which $\forall v \in V$:
$1 \leqslant indegree(v) + outdegree(v) \leqslant 2$. Since y_0 has indegree =0 and outdegree =1. Then, there exists another node y' s.t.
$indegree(y') + outdegree(y') = 1$

PPAD complete problems

1. Finding the Nash equilibrium on a 2-player game
2. Finding a three-colored point in Sperner's Lemma
3. Brouwer fixed point theorem

References

- D. S. Johnson, C. H. Papadimitriou, and M. Yannakakis
  How easy is local search?
- C.H. Papadimitriou
  Computational Complexity
- C.H. Papadimitriou
  On the complexity of the parity argument and other inefficient proofs of existence
  Journal of Computer and System Sciences , pages 498-532, 1994.
- M. Yannakakis.
  Survey: Equilibria, fixed points, and complexity classes.