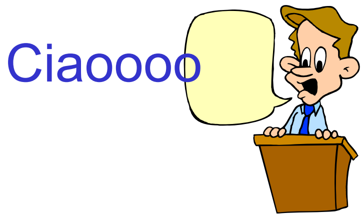
CiE 2023: informal talk

# Complexity Classification of Complex-Weighted Counting Acyclic Constraint Satisfaction Problems

July 28, 2023. 11:00-11:30 (GET). Batumi, Georgia (Online)

Dr. Tomoyuki Yamakami
University of Fukui, Fukui, JAPAN




# Synopsis of Today's Talk

- This seminal talk concerns
  - counting acyclic constraint satisfaction problems (or #ACSPs).
- I will try to
  - develop a proof technique to cope with #ACSPs.
- I will present
  - two complete classifications of $\mathbb{C}$-valued #ACSPs.

- YouTube Search ↪ Tomoyuki Yamakami
- twitter ↪ tomoyamakami

# I. Counting CSPs

1. CSPs
2. #CSPs
3. Acyclic CSPs
4. Complexity class LOGCFL
5. Acyclicity
6. Quick Examples
7. Constraint hypergraphs
8. #ACSPs

# Constraint Satisfaction Problems (CSPs)

- Our subject is constraint satisfaction problems (or CSPs).
- CSPs with Boolean domains are briefly called Boolean CSPs.
- Typical Boolean CSPs include 3SAT.
- Schaefer (1978) considered CSPs with Boolean domains and proved the dichotomy theorem (or the dichotomy classification) for them.
- (Claim) Any CSP with Boolean domains is either in P or NP-complete.
- In the rest of this talk, we are focused on Boolean CSPs.

# Counting CSPs (#CSPs)

- As a variant of CSPs, we focus on counting (Boolean) CSPs (or succinctly, #CSPs).
- Creignou and Herman (1996) proved a complete classification of #CSPs with {0,1}-valued constraint functions (or unweighted #CSPs).
- Dyer, Goldberg, and Jerrum (2009) presented a classification for nonnegative real weighted #CSPs.
- Cai, Lu, and Xia (2014) obtained a classification for complex-weighted #CSPs.
- Dyer, Goldberg, and Jerrum (2010) studied randomized approximate counting.
- Yamakami (2012) gave a randomized approximation classification for complex-weighted #CSPs.

# Acyclic CSPs

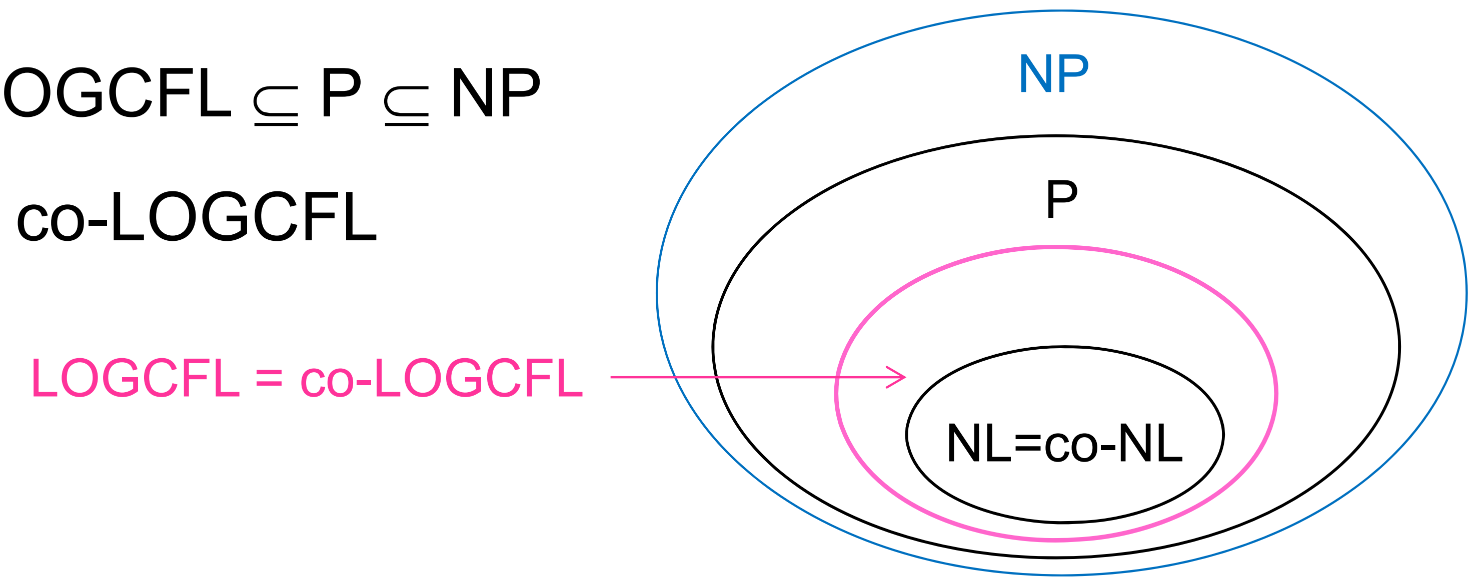
- Gottlob, Leone, and Scarcello (2001) studied the acyclic version of CSPs, called ACSPs, in connection to database theory.
- They proved that the generic problem ACSP (not necessarily limited to Boolean) is complete for LOGCFL.

- In the next two slides, we will see the precise definitions of “LOGCFL” and “acyclicity”.

# Complexity Class LOGCFL

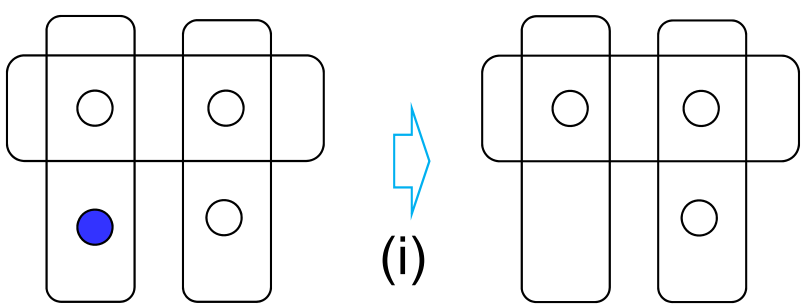
- A decision problem (or equivalently, a language) L is in LOGCFL if there is a two-way auxiliary pushdown automaton (or an aux-2npda) M such that, for any input x,
  1. $x \in L \leftrightarrow$ there exists an accepting computation path of M on x (or x is accepted by M), and
  2. M runs in polynomial time using logarithmic work space (or log space) on all inputs.

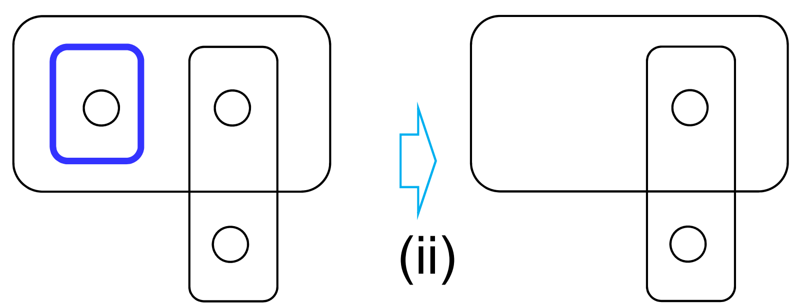
- $L \subseteq NL \subseteq LOGCFL \subseteq P \subseteq NP$
- LOGCFL = co-LOGCFL



# Acyclicity (or $\alpha$-Acyclicity)

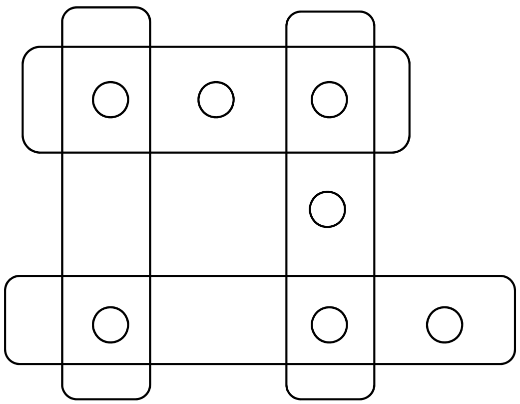
- A hypergraph G is of the form (V,E) with a finite set V of vertices and a set E of hyperedges (i.e., subsets of V).
- The empty hypergraph has no vertex.
- A hypergraph G is acyclic $\Leftrightarrow$ after applying the following actions (i)-(ii) finitely many times, G becomes the empty hypergraph.
  i. Remove vertices that appear in at most one hyperedge.
  ii. Remove hyperedges that are either empty or contained in other hyperedges.



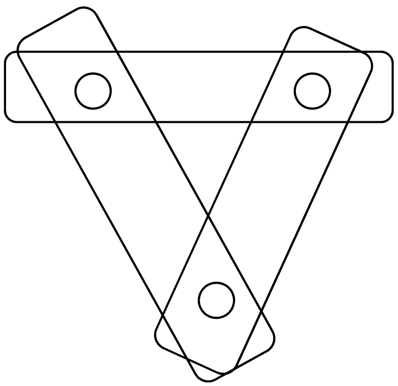



# Quick Examples

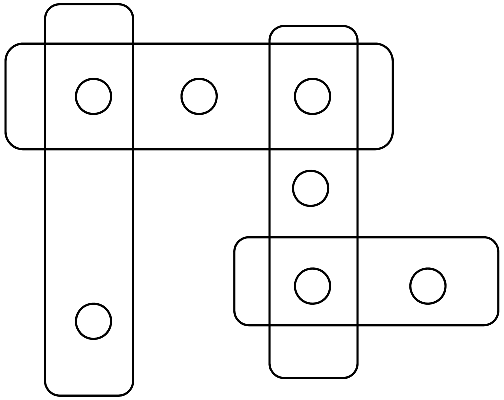
cyclic hypergraph



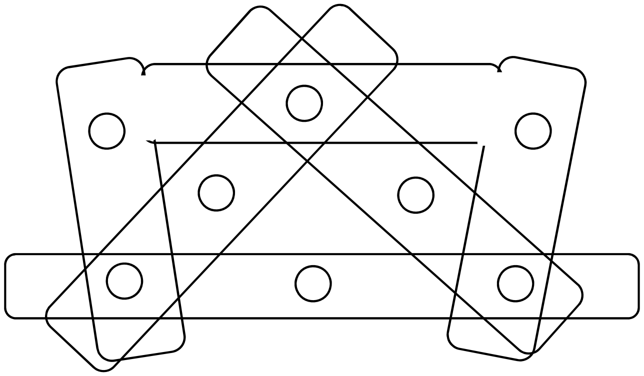
cyclic hypergraph



acyclic hypergraph



acyclic hypergraph



# Constraint Hypergraphs

- Consider a #CSP instance $I = (Var, C)$, where $Var = \{ v_i \}_{i \in [t]}$ is a set of Boolean variables and $C = \{ C_i \}_{i \in [s]}$ is a set of $\mathbb{C}$-valued constraints of the form

  $$C_i = ( f_i, (v_{i1}, v_{i2}, \ldots, v_{ik}) )$$

  for any $i \in [s]$.
- We associate it with a labeled hypergraph $G_I = (V_I, E_I)$, where
  - $V_I = Var$, and
  - $E_I = \{ \{ v_1, v_2, \ldots, v_k \} \mid (f, (v_1, v_2, \ldots, v_k)) \in C \}$ whose hyperedge $\{ v_1, v_2, \ldots, v_k \}$ has $f$ as its label.
- We call $G_I$ the constraint hypergraph of $I$.
- A #CSP instance $I$ is acyclic $\Leftrightarrow$ $G_I$ is acyclic

# #ACSPs

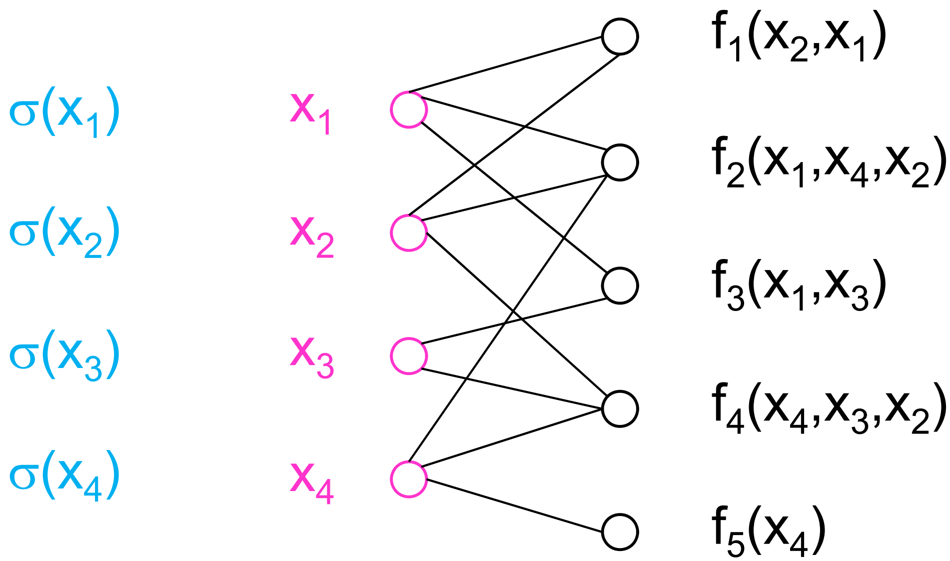
- Let F be any set of $\mathbb{C}$-valued constraint functions with Boolean domains.

- F-restricted counting acyclic constraint satisfaction problem (or #ACSP(F))
  - instance: $I = (Var, C)$ with a set $Var = \{ v_i \}_{i \in [t]}$ of Boolean variables and a set $C = \{ C_i \}_{i \in [s]}$ of $\mathbb{C}$-valued constraints $C_i = ( f_i, (v_{i1}, v_{i2}, \ldots, v_{ik}) )$ s.t. $f_i \in F \cup \{ \Delta_0, \Delta_1 \}$ for any $i \in [s]$ and $I$ is acyclic
  - output: $count(I) = \sum_\sigma \prod_{i \in [s]} f_i(\sigma(v_{i1}), \sigma(v_{i2}), \ldots, \sigma(v_{ik}))$, where $\sigma : Var \to \{0,1\}$

A #ACSP instance: I = (Var, C) with

Var = { $x_1$, $x_2$, $x_3$, $x_4$ } and

C = { ($f_1$,($x_2$,$x_1$)), ($f_2$,($x_1$,$x_4$,$x_2$)), ($f_3$,($x_1$,$x_3$)), ($f_4$,($x_4$,$x_3$,$x_2$)), ($f_5$,($x_4$)) }




$\sigma: \{x_1, x_2, x_3, x_4\} \to \{0,1\}$

$$count(I) = \sum_{\sigma} f_1(\sigma(x_2),\sigma(x_1)) f_2(\sigma(x_1),\sigma(x_4),\sigma(x_2)) f_3(\sigma(x_1),\sigma(x_3))\, f_4(\sigma(x_4),\sigma(x_3),\sigma(x_2)) f_5(\sigma(x_4))$$

# II. #LOGCFL and #LOGCFL$_{\mathbb{C}}$

1. #LOGCFL and #LOGCFL$_{\mathbb{C}}$
2. Examples
3. Logspace reductions
4. #LOGCFL-completeness

# #LOGCFL and #LOGCFL$_{\mathbb{C}}$

- We discuss a counting version of LOGCFL.
- #LOGCFL consists of all counting problems f that satisfy the following condition:
  - there are an aux-2npda M s.t., for any x, f(x) equals the total number of accepting paths of M on the input string x.
- We can expand #LOGCFL to #LOGCFL$_{\mathbb{C}}$ by treating complex numbers as individual “symbolic” objects.
- This is a common way of defining P$_{\mathbb{C}}$, NP$_{\mathbb{C}}$, FP$_{\mathbb{C}}$, and #P$_{\mathbb{C}}$ induced directly from P, NP, FP, and #P.

❑ Refer to, e.g., Arora-Barak’s textbook (Computational Complexity, 2009).

# Examples

- We see a few examples of #LOGCFL problems.

- Ranking of 1dpda problem (or

  The number of strings in L(M) that are lexicographically smaller than x

  - instance: a one-way deterministic pushdown automaton (or a 1dpda) M and an input string $x \in \{0,1\}^*$.
  - output: the rank of x in L(M).

- Counting $SAC^1$ problem (or #SAC1P)
  - instance: an encoding $\langle C \rangle$ of a leveled semi-unbounded Boolean circuit of size at most n and of depth at most log(n) with n input bits and an input string $x \in \{0,1\}^n$.
  - output: the total number of accepting computation subtrees of C on the input x.

# Logspace Reductions

- The logarithmic-space reducibility is commonly used for the NL-completeness of languages.
- We expand it to reductions between functions.

- Let f,g be any two functions.
- f is logspace reducible to g ( $f \leq^{L} g$ ) $\Leftrightarrow$

  $\exists\, h \in$ FL (polynomially bounded) $\forall x \in \Sigma^{*}$ [ $f(x) = g(h(x))$ ]

- A function f is #LOGCFL-hard (under logspace reductions) $\Leftrightarrow$ $\forall g \in$#LOGCFL [ $g \leq^{L} f$ ].

- A function f is #LOGCFL-complete (under logspace reductions) $\Leftrightarrow$ f is #LOGCFL-hard and f is in #LOGCFL.

# #LOGCFL-Completeness

- Lemma
  1) #SAC1P is #LOGCFL-complete.
  2) $RANK_{1dpda}$ is #LOGCFL-complete. (Vinay (1991))

# III. Various Constraint Functions

1. How to express constraint functions
2. Counting acyclic 2CNF satisfiability problem
3. ED, NZ, and IM
4. Useful facts

# How to Express Constraint Functions I

- We assume the standard lexicographic order on $\{0,1\}^k$.
- Let f: $\{0,1\}^k \rightarrow \mathbb{C}$ be any constraint function.
- We express this f as a k-tuple $(f(0^k),f(0^{k-1}1),\ldots,f(1^k))$.
  - If k=1, then f is expressed as (f(0),f(1)).
  - If k=2, then f is expressed as (f(00),f(01),f(10),f(11)).
- f is symmetric $\Leftrightarrow$ $\forall\pi:[k]\rightarrow[k]$ permutation $\forall x_1,x_2,..,x_k\in\{0,1\}$ [ $f(x_1,x_2,\ldots,x_k) = f(x_{\pi(1)},x_{\pi(2)},\ldots,x_{\pi(k)})$ ]
- For a symmetric constraint function f, f is expressed as $[a_0,a_1,a_2,\ldots,a_k]$, where $a_i = f(x)$ for any $x\in\{0,1\}^k$ containing exactly i 1s.
- E.g., consider f(x) = the number of 1s in x (mod 2).
  - f = ( 0,1,1,0,1,0,0,1 ) and f = [ 0,1,0,1 ]

# How to Express Constraint Functions II

- Examples
  - $AND_k$ = [0,0,0,…,0,1] (k zeros)
  - $OR_k$ = [0,1,1,…,1] (k ones)
  - $NAND_k$ = [1,1,…,1,0] (k ones)
  - $EQ_k$ = [1,0,0,…,0,1] (k-1 zeros)
  - XOR = $NEQ_2$ = [0,1,0]
  - Implies = (1,1,0,1) "$x \rightarrow y$"
  - RImplies = (1,0,1,1) "reverse implies: $y \rightarrow x$"
  - $\Delta_0$ = [1,0] and $\Delta_1$ = [0,1] (special unary functions)
- Equalities
  - XOR(x,y) = $OR_2$(x,y)$NAND_2$(x,y)
  - $EQ_2$(x,y) = Implies(x,y)Rimplies(x,y)

# Counting Acyclic 2CNF Satisfiability Problem

- A Boolean formula $\varphi$ is acyclic $\Leftrightarrow$ its associated constraint hypergraph $G_\varphi$ is ac

2CNF: $\varphi \equiv (x_1 \vee x_2) \wedge (x_1 \vee \neg x_3) \wedge (\neg x_2 \vee x_3)$

- Counting acyclic 2CNF satisfiability problem (or #Acyc-2SAT)
  - instance: an acyclic 2CNF Boolean formula $\varphi$
  - output: the total number of satisfying assignments of $\varphi$

#ACSP$^{(-)}$(F) means that unary constraints in use are limited to [0,1], [1,0], [0,0], and [1,1].

Implies = (1,1,0,1)

Lemma

1) #Acyc-2SAT $\leq^{L}$ #ACSP(Implies)
2) #ACSP$^{(-)}$(Implies) $\leq^{L}$ #Acyc-2SAT

# ED, NZ, and IM

- We define three important sets of constraint functions.
- ED = the set of all constraint functions that are products of some of the following functions:

  unary functions, $EQ_2$, and XOR.
- NZ = the set of all non-zero constraint functions.
- IM = the set of all constraint functions, not in NZ, which are products of some of unary functions and “Implies”.
- Examples
  - $AND_2 \in ED$, because $AND_2(x,y) = EQ_2(x,y)\Delta_1(x)$
  - $EQ_3 \in ED$, because $EQ_3(x,y,z) = EQ_2(x,y)EQ_2(y,z)$
  - $EQ_2 \in IM$, because $EQ_2(x,y) = Implies(x,y)Implies(y,x)$

# Useful Facts

- We can prove the following statements.

1) $\#SAC1P \leq^{L} \#ACSP(OR_2,XOR)$
2) $\forall F \subseteq ED$ [ $\#ACSP(F)$ is in $FL_{\mathbb{C}}$ ]
3) $\forall f \notin ED$ [ $\#ACSP(OR_2) \leq^{L} \#ACSP(f)$ ]
4) $\forall f \notin IM \cup ED$ [ $\#ACSP(OR_2,XOR) \leq^{L} \#ACSP(f)$ ]

- Theorem

  For any constraint set F, #ACSP(F) is in $\#LOGCFL_{\mathbb{C}}$.

# IV. Two Classification Results

1. Trichotomy classification
2. Dichotomy classification
3. Acyclic T-constructibility

# Trichotomy Classification

- We allow the free use of unary constraints as part of inputs.
- We then obtain the following trichotomy classification of #ACSPs.

- Under the free use of unary constraints, given any #ACSP f, the following statements hold.
  1) If all constraint functions of f are in ED, then f is in $FL_{\mathbb{C}}$.
  2) Otherwise, if all constraints of f are in IM, then f is in #Acyc-2SAT-hard.
  3) Otherwise, f is #LOGCFL-hard.

# Dichotomy Classification

- Next, we consider the case where the free use of XOR is allowed together with unary constraints.
- In this particular case, we can obtain the following dichotomy classification of #ACSPs.

- Under the free use of XOR and unary constraints, given any #ACSP f, the following statements hold.
  1) If all constraint functions of f are in ED, then f is in $FL_{\mathbb{C}}$.
  2) Otherwise, f is #LOGCFL-hard.

# Acyclic T-Constructibility

- To prove the aforementioned classification results, we need to develop a crucial technical tool, called acyclic T-constructibility or AT-constructibility.
- This is an adaptation of T-constructibility notion introduced by Yamakami (2012, I&C).

- Due to the time constraint, we omit the detailed description of AT-constructibility in this talk.

# V. Open Problems

1. Open problems

# Open Problems

- Numerous questions have left unsolved in this study.
- We list a few such questions below.

1. Find a complete classification of #ACSPs when we place a restriction on the choice of weight types (such as nonnegative real numbers).
2. Find a randomized approximate classification of #ACSPs.
3. What is the exact complexity of #Acyc-2SAT?
4. Is it true that #L $\neq$ #LOGCFL or even FL $\neq$ #LOGCFL?

*Thank you for listening*

# Q & A

I’m happy to take your question!

END