Enumeration of Circuits and Minimal Forbidden Sets

Frederik Stork

ILOG GmbH, Germany

Marc Uetz

Universiteit Maastricht, Netherlands

Independence Systems

Given ground set V, $|V| = n$, then $\mathcal{I} \subseteq 2^V$ is independence system if

$$W \subseteq U \in \mathcal{I} \;\Rightarrow\; W \in \mathcal{I}$$

Bases $\mathcal{B}$: $B \in \mathcal{I}$ $\subseteq$-maximal
Circuits $\mathcal{C}$: $C \notin \mathcal{I}$ $\subseteq$-minimal

Example. $\mathcal{I}$ = forests in a graph: bases = spanning trees, circuits = simple cycles

Given membership oracle for $\mathcal{I}$, we want to enumerate all circuits of $\mathcal{I}$.

Complexity of Enumeration Problems

Given V, and an implicit, say poly(n) description of $\mathcal{C} \subseteq 2^V$.

- Total polynomial time: Enumeration of $\mathcal{C}$ in time poly$(n, |\mathcal{C}|)$
- Incremental polynomial time: Given $\mathcal{X} \subseteq \mathcal{C}$, either compute $U \in \mathcal{C} \setminus \mathcal{X}$ or decide $\mathcal{X} = \mathcal{C}$ in time poly$(n, |\mathcal{X}|)$

[Valiant'79, Johnson, Yannakakis, Papadimitriou'88]

Maximal Independent Set Enumeration is Hard

THEOREM. Unless P=NP, no algorithm exists that enumerates the bases $\mathcal{B}$ of any independence system $\mathcal{I}$ in total polynomial time.

[Lawler, Lenstra, Rinnooy Kan '80]

Proof. Reduction from SATISFIABILITY. □

Likewise. . .

THEOREM. Unless P=NP, no algorithm exists that enumerates the circuits $\mathcal{C}$ of any independence system $\mathcal{I}$ in total polynomial time.

Proof. Given $\mathcal{I}$, suppose $\mathcal{A}$ is such an algorithm. Define $\mathcal{I}^D$ by

$$W \in \mathcal{I}^D :\Leftrightarrow V \setminus W \notin \mathcal{I}\,.$$

U max. indep. for $\mathcal{I}^D$ $\Leftrightarrow$ $V \setminus U$ min. cover for $\mathcal{I}$. Then $\mathcal{A}$ enumerates max. indep. sets of any $\mathcal{I}^D$

□

Question. Particular independence systems?

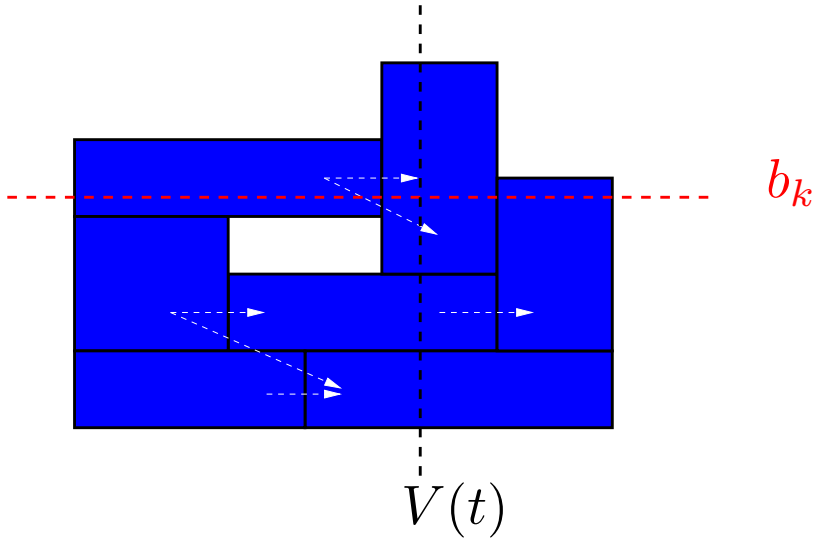
Motivation: Resource Constrained Scheduling

i, j jobs $\in V := \{1, ..., n\}$

$\prec$ partial order on V
(precedence constraints)

b_k availability of resource k

a_{kj} resource k as required by job j



Resource constraints: $\sum_{j \in V(t)} a_{kj} \leq b_k$ for all k and t

Circuits = Minimal Forbidden Sets

trivial: $F = \{i, j\}$ with $i \prec j$, or

non-trivial:

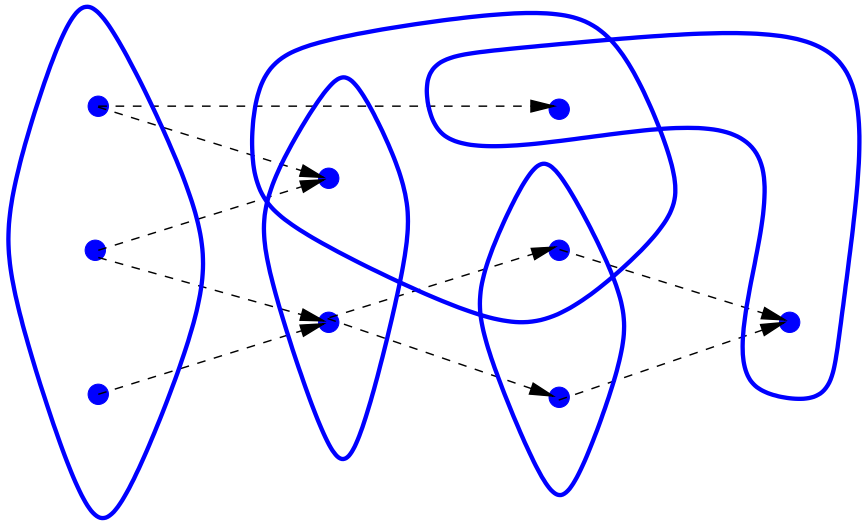
- F anti-chain for poset $(V, \prec)$
- $\sum_{j \in F} a_{kj} > b_k$ for some k
- F $\subseteq$–minimal

Required for:

- policies in stochastic scheduling
- cutting planes for IP formulations
- $\cdots$

Independence System Hypergraph

- nodes V = jobs
- hyperedges $\mathcal{C}$ = min. forb. sets



jobs $U \subseteq V$ feasible $\Leftrightarrow$ U independent set in $(V, \mathcal{C})$

Problem: List of all (non-trivial) minimal forbidden sets $\mathcal{C}$ required

But: The (non-trivial) minimal forbidden sets $\mathcal{C}$ not given explicitly

Given Explicitly: Linear System $Ax \leq b$

$$
\begin{array}{c}
 \\
i \prec j
\end{array}
\begin{array}{c}
\begin{array}{cccc} & i & & j \end{array} \\
\begin{pmatrix}
a_{11} & \cdots & a_{1n} \\
a_{21} & \cdots & a_{2n} \\
\vdots & & \vdots \\
a_{k1} & \cdots & a_{kn} \\
 & \cdots & \\
 1 & & 1 \\
 & \cdots &
\end{pmatrix}
\end{array}
\begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{pmatrix}
\leq
\begin{pmatrix} b_1 \\ b_2 \\ \vdots \\ b_k \\ 1 \\ \vdots \\ 1 \end{pmatrix}
$$

jobs $U \subseteq V$ feasible

$\Leftrightarrow$

$Ax(U) \leq b$

Minimally infeasible vectors $x \in \{0,1\}^n$ = minimal forbidden sets F

The Enumeration Problem

Assume $A \geq 0$, $b \geq 0$, integral.

INPUT:	Linear inequality system $Ax \leq b$
OUTPUT:	All circuits: minimal infeasible $\{0,1\}$–vectors x

Poly-Time Enumeration is Hard

THEOREM. Unless P=NP, no total poly-time algorithm exists for the enumeration problem.

Proof. Take graph $G = (V, E)$, decision problem $\exists$ indep. set $\geq t$? Define

$$\begin{array}{c} \\ \{k,j\} \\ \{i,j\} \\ \\ \\ \end{array} \begin{array}{c} \begin{array}{ccccc} k & & i & & j \end{array} \\ \begin{pmatrix} & & \cdots & & \\ 1 & & & 1 & \\ & 1 & & 1 & \\ & & \cdots & & \\ 1 & 1 & 1 & 1 & 1 \end{pmatrix} \end{array} \begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{pmatrix} \leq \begin{pmatrix} 1 \\ 1 \\ \vdots \\ 1 \\ t-1 \end{pmatrix}$$

Assume total poly-time algorithm $\mathcal{A}$, with $p(n, |\mathcal{C}|)$. Let $\mathcal{A}$ run for $p(n, |E|) + 1$ time.

1. $\mathcal{A}$ outputs min. infeas. x with $\geq t$ one's $\rightarrow G$ 'Yes'
2. $\mathcal{A}$ terminates w/o min. infeas. x with $\geq t$ one's $\rightarrow G$ 'No'
3. $\mathcal{A}$ outputs no min. infeas. x with $\geq t$ one's, and doesn't terminate $\rightarrow G$ 'Yes'

. . . hence $\mathcal{A}$ decides NP-hard INDEP.SET $\square$

Poly-Time Enumeration for the Dual is easier

The 'Dual':

INPUT: Linear inequality system $Ax \leq b$
OUTPUT: All maximal feasible $\{0,1\}$–vectors x

THEOREM. There is an incremental (quasi-)poly-time algorithm [order $t^{o(\log t)}$]
[Boros, Elbassoni, Gurvich, Khachiyan, Makino '02]

Proof idea. $\#$ min. infeas. $x \leq nm \; \#$ max. feas. x, so generate both sets, solved by reduction to (generalized) hypergraph dualization [Fredman & Khachiyan '96] □

Note. Such an algorithm is not likely to exist to for min. infeasible x for $Ax \leq b$.

The Counting Problem

INPUT:	Linear inequality system $Ax \leq b$
OUTPUT:	# of min. infeasible $\{0,1\}$–vectors x

Counting is Hard

THEOREM. The counting problem is #P-complete.

Proof. (#P-hardness) Given a poset $(V, \prec)$. Consider MAXAC: Compute maximum cardinality anti-chain

- MAXAC is #P-complete
  [Provan & Ball '83]
- max. cardinality t in poly(n)
  [Max-Flow reduction]

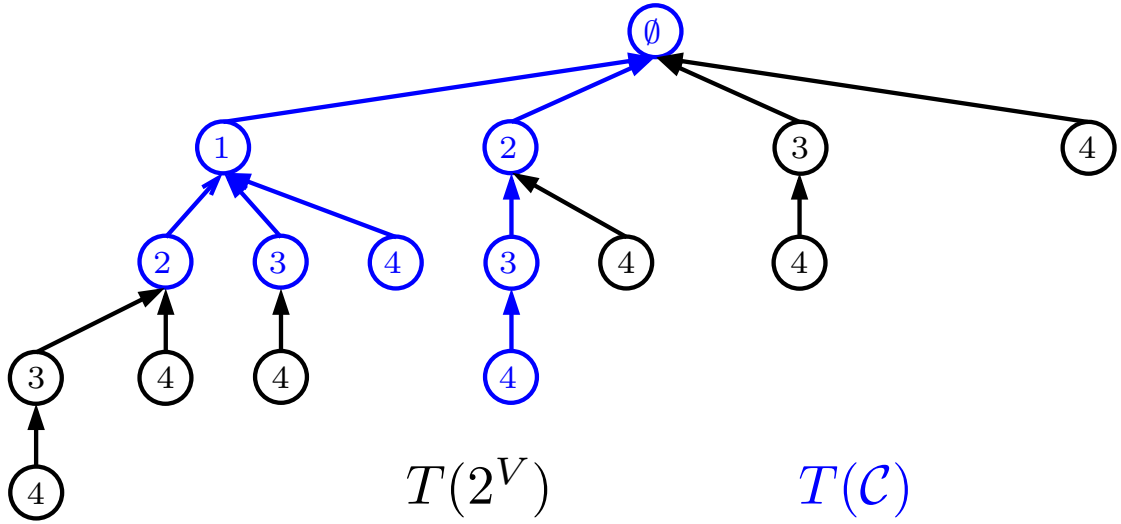
$$\begin{array}{c} \\ k \prec j \\ i \prec j \\ \\ \\ \end{array} \begin{array}{c} \begin{array}{ccccc} k & i & & j & \end{array} \\ \begin{pmatrix} & & \cdots & & \\ 1 & & & 1 & \\ & 1 & & 1 & \\ & & \cdots & & \\ 1 & 1 & 1 & 1 & 1 \end{pmatrix} \end{array} \begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{pmatrix} \leq \begin{pmatrix} 1 \\ 1 \\ \vdots \\ 1 \\ t-1 \end{pmatrix}$$

Then: # of min. infeasible $\{0,1\}$–vectors x
$= |\prec| +$ # of MAXAC's

□

Implementation of Enumeration Algorithm

$$\begin{pmatrix} 3 & 2 & 1 & 1 \\ 1 & 1 & 0 & 0 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{pmatrix} \leq \begin{pmatrix} 3 \\ 1 \end{pmatrix}. \quad \mathcal{C} = \{\{1,2\},\{1,3\},\{1,4\},\{2,3,4\}\}$$



$\mathcal{C}$ = leaves of tree $T(\mathcal{C})$ (several heuristics to fathom the tree)

Empirical: Comparison to Divide & Conquer

1. for any row k of $Ax \leq b$, compute $\mathcal{C}_k$ (incr. poly time)

2. store only $\subseteq$–minimal sets of $\bigcup \mathcal{C}_k$

[Lawler, Lensta, Rinnooy Kan '80, Bartusch '84]

	∅ CPU (sec.)	max. CPU (sec.)
Divide & Conquer	2.19	102.0
Computation of $T(\mathcal{C})$	0.01	0.2

480 scheduling instances with 30 jobs and 4 resources

Analysis: Special cases

Scheduling application with one resource type

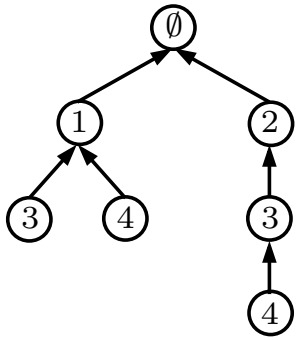
$$\begin{array}{c} \\ k \prec j \\ i \prec j \\ \\ \end{array} \begin{array}{c} \begin{array}{ccccc} k & & i & & j \end{array} \\ \begin{pmatrix} & & \cdots & & \\ 1 & & & & 1 \\ & 1 & & & 1 \\ & & \cdots & & \\ a_1 & \geq & \cdots & \geq & a_n \end{pmatrix} \end{array} \begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{pmatrix} \leq \begin{pmatrix} 1 \\ 1 \\ \vdots \\ 1 \\ b \end{pmatrix}$$

Task: Enumerate all min. infeas. $x \in \{0,1\}^n$

THEOREM. Our enumeration algorithm can be implemented to do this in incremental polynomial time.

Analysis [contd.]

Proof idea.



$T(\mathcal{C})$

1) Any infeas. set in $T(\mathcal{C})$ is min. infeasible.
2) At any node, poly(n) test decides if infeasible descendant exists (max. weight anti-chain!)
3) put all together. . .

□

Discussion

- Surprising: minimal infeasible x harder than maximal feasible x for $Ax \leq b$
- Related: (generalized) dualization of hypergraphs

[e.g., Endre Boros et. al.]

- Open: $\exists$ Poly-time enumeration algorithm for general scheduling problem?

Complexity of Counting Problems

Given V and an implicit description of $\mathcal{C} \subseteq 2^V$, $|\mathcal{C}| = c$.

- Membership in #P: c can be determined by counting the accepting computations of a nondet. poly-time Turing machine

- #P-hardness: Computation of c yields a solution for any problem in #P (typically: parsimonious reduction)

- #P-complete problems: Counting hamiltonian cycles or perfect matchings in a bipartite graph