Upper Bounding the Performance of Arbitrary Finite LDPC Codes on Binary Erasure Channels

An efficient exhaustion algorithm for error-prone patterns

C.-C. Wang, S.R. Kulkarni, and H.V. Poor

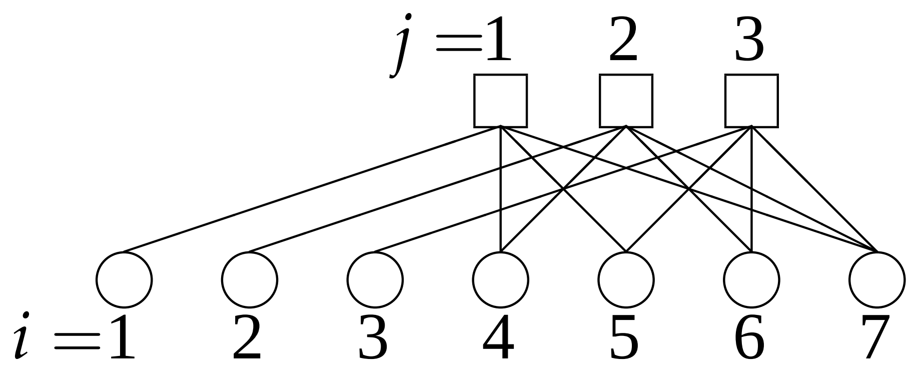
School of Electrical & Computer Engineering, Purdue Unversity
Department of Electrical Engineering, Princeton University

Content

- Brief introduction
  - Stopping sets in erasure channels. Hardness & Applications.
  - Existing approaches for upper / lower bounding the fixed code performance:
- A tree-based upper bound for bit error rates.
- Frame error rates and trapping sets.

Stopping Sets (SSs)

- Definition: a set of variable nodes such that the induced graph contains no check node of degree 1.
- Example: The binary (7,4) Hamming Code

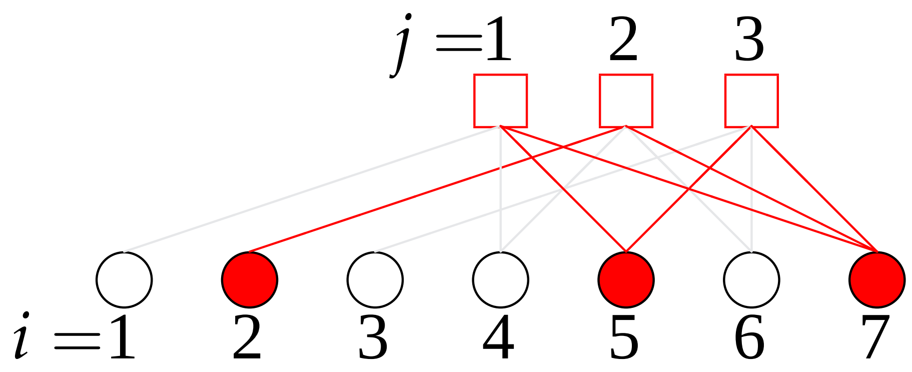


$(2,5,7)$ a valid codeword vs. $(4,6,7)$ a pure stopping set.

Hamming distance vs. Stopping distance

Stopping Sets (SSs)

- Definition: a set of variable nodes such that the induced graph contains no check node of degree 1.
- Example: The binary (7,4) Hamming Code

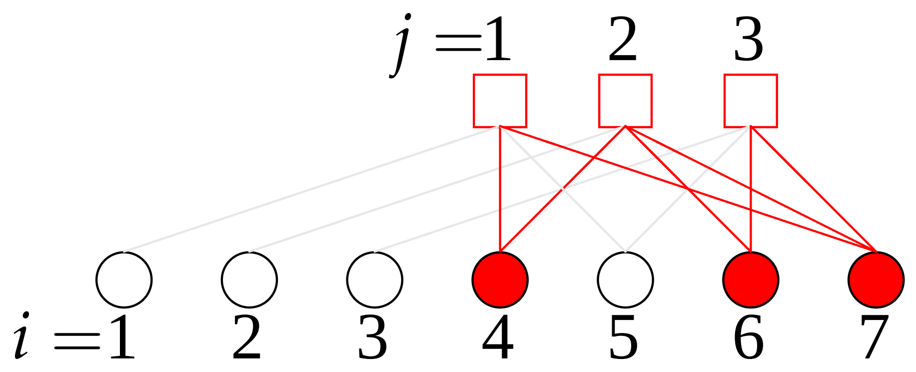


$(2,5,7)$ a valid codeword vs. $(4,6,7)$ a pure stopping set.

Hamming distance vs. Stopping distance

Stopping Sets (SSs)

- Definition: a set of variable nodes such that the induced graph contains no check node of degree 1.
- Example: The binary (7,4) Hamming Code

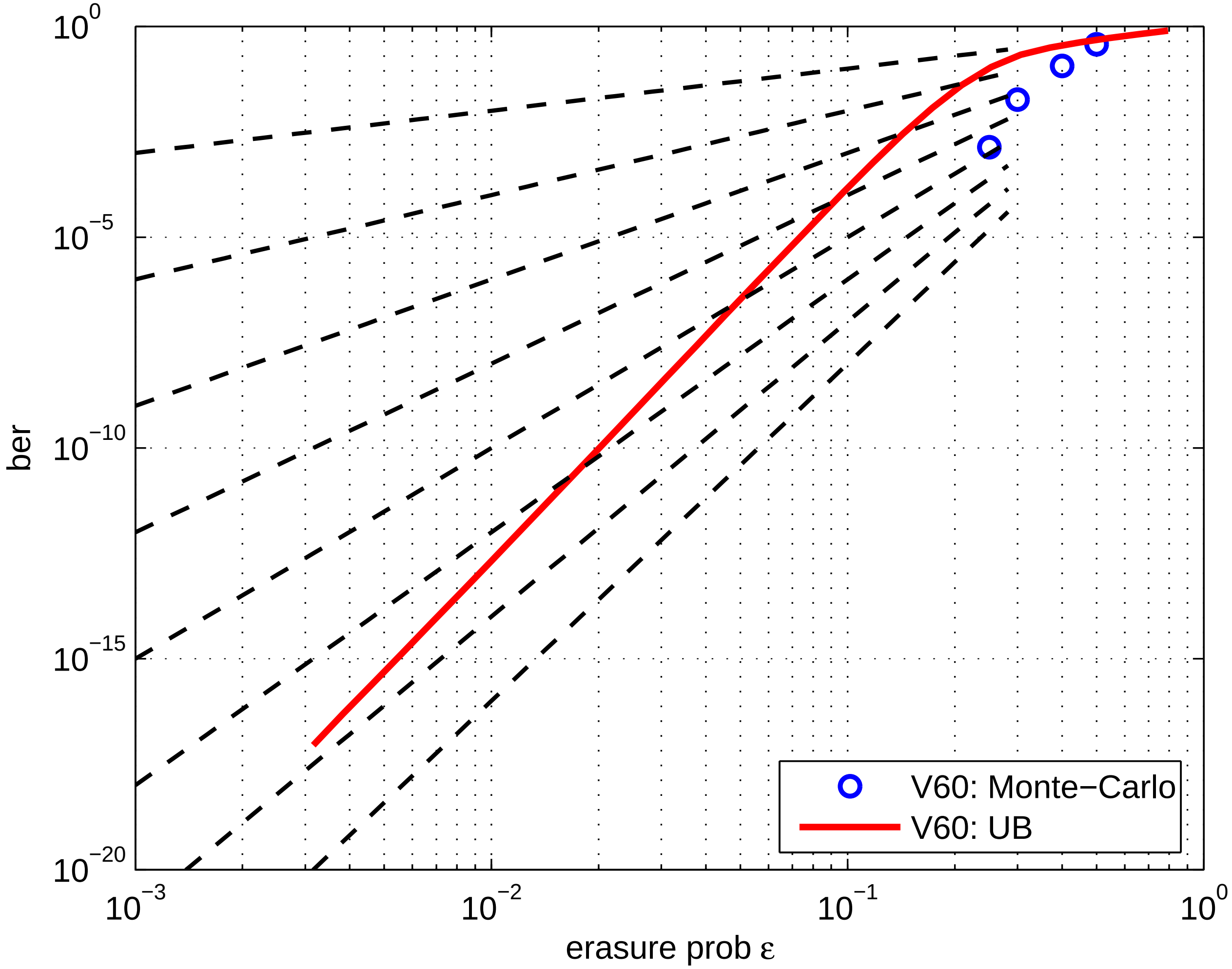


$(2,5,7)$ a valid codeword vs. $(4,6,7)$ a pure stopping set.

Hamming distance vs. Stopping distance

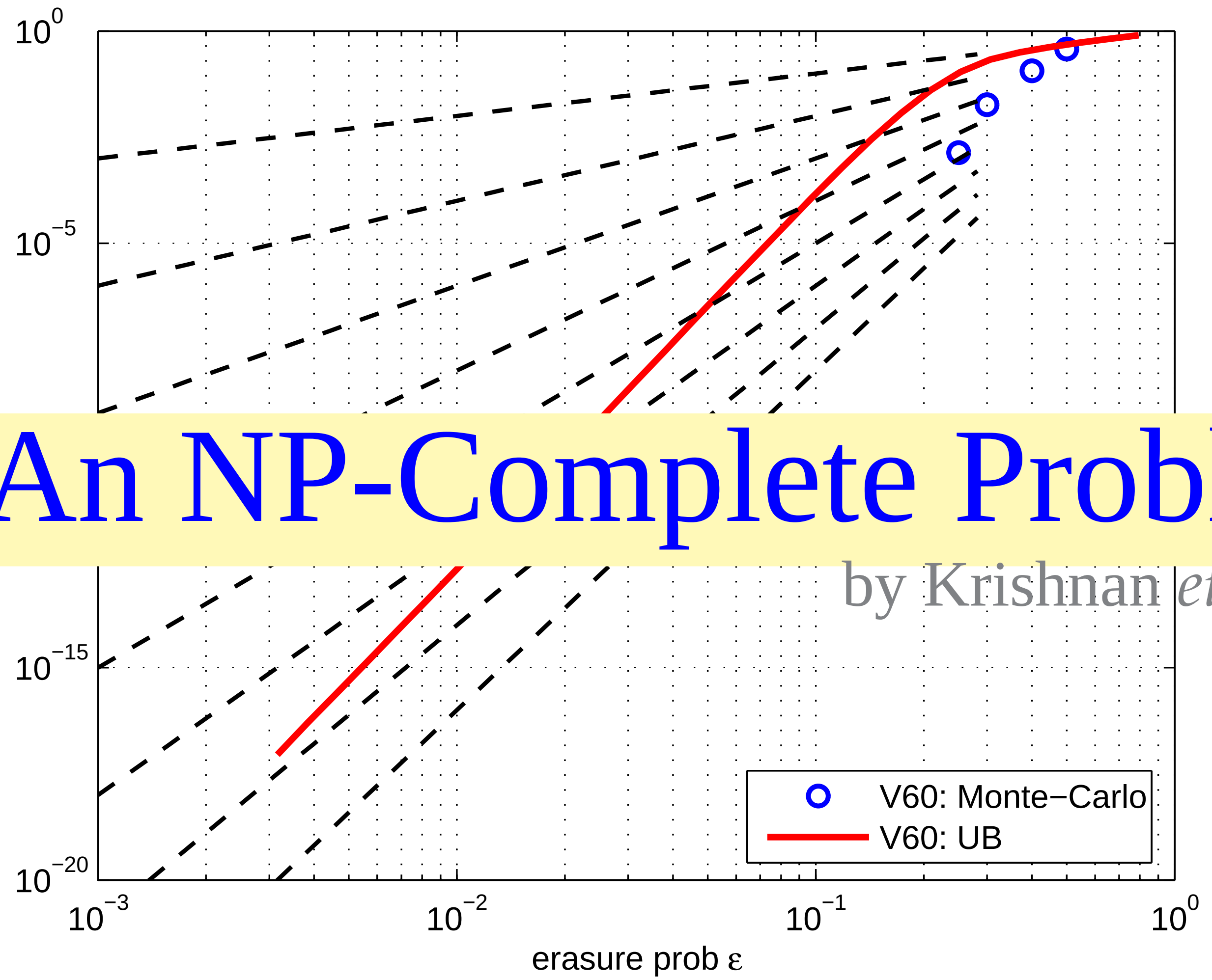
Upper Bounds vs. Exhausting Minimum Stopping Sets

An $n = 72$ regular (3,6) LDPC code



Upper Bounds vs. Exhausting Minimum Stopping Sets

An $n = 72$ regular (3,6) LDPC code

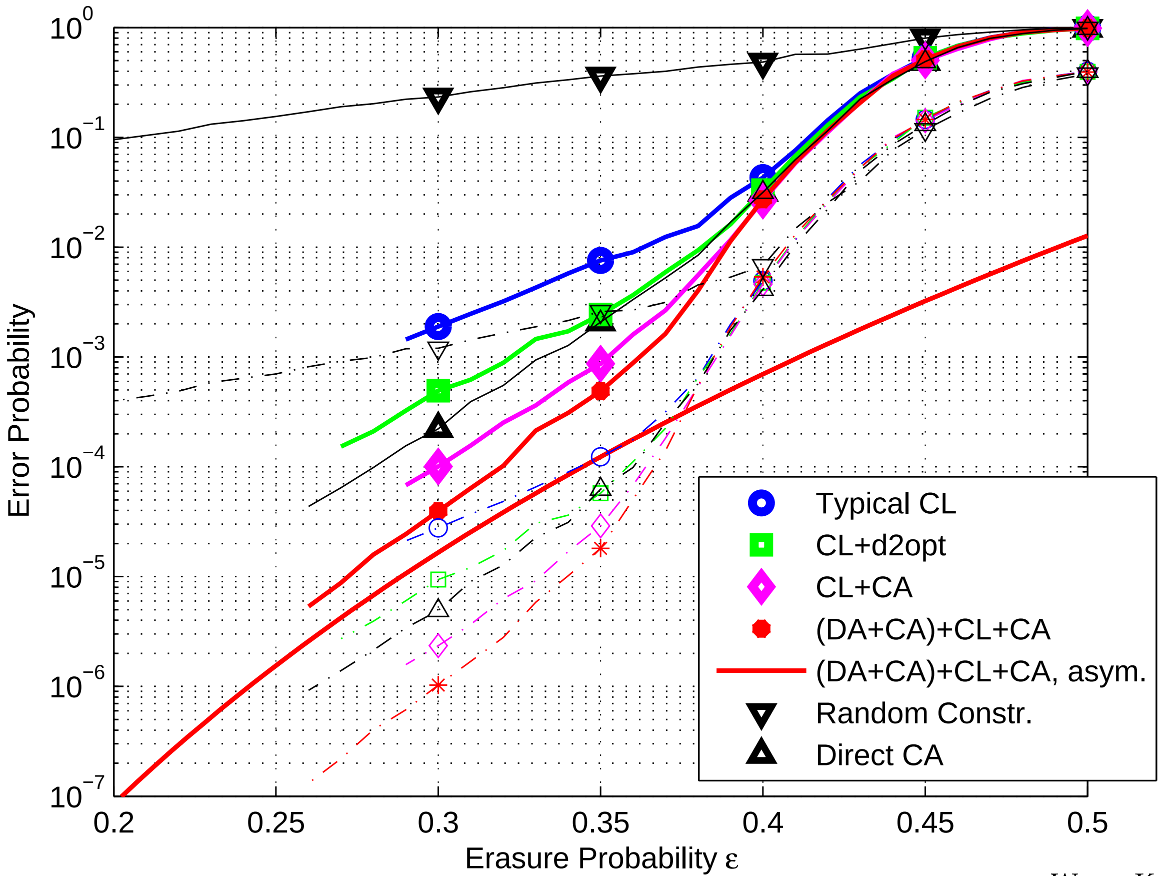


An NP-Complete Problem !!

by Krishnan *et al.*

A Hard but Useful Problem

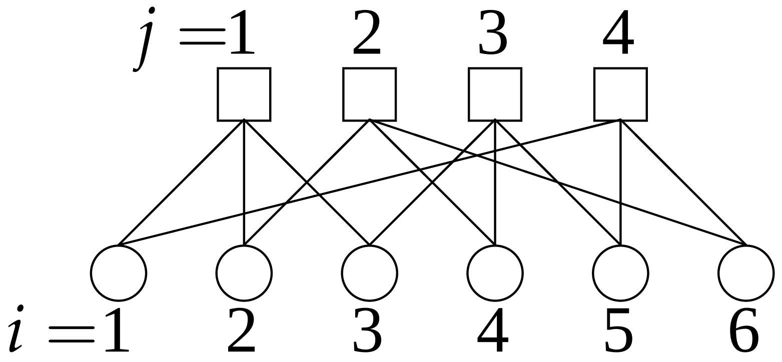
- As an upper bound: Guaranteed worst performance for extremely low ber regimes.
- As an SS exhaustion algorithm: Code annealing [Wang 06].



Existing Approaches

- Ensemble analysis: [Di *et al.* 02] avg. ber and FER curves, [Amraoui *et al.* 04] the scaling law for water fall regions.
- Fixed Code analysis: [Holzlöhner *et al.* 05] dual adaptive importance sampling, [Stepanov *et al.* 06] instanton method
  - Enumerating bad patterns: [Richardson 03] error floors of LDPC codes, [Yedidia *et al.* 01] projection algebra, [Hu *et al.* 04] the minimum distance of LDPC codes.
  - They are all inexhaustive enumeration.

The Bit-Oriented Detection

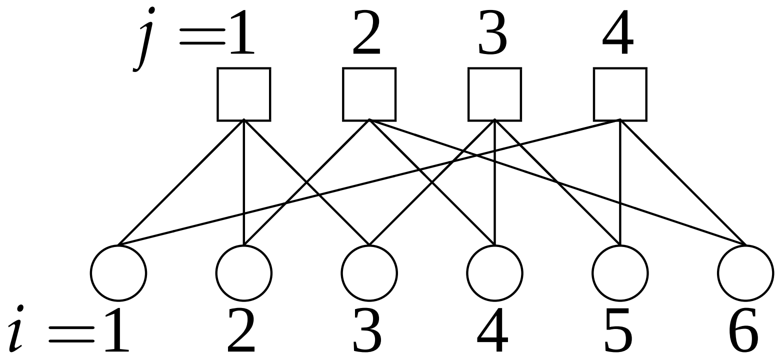


Using

1: for erasure

0: for non-erasure

The Bit-Oriented Detection



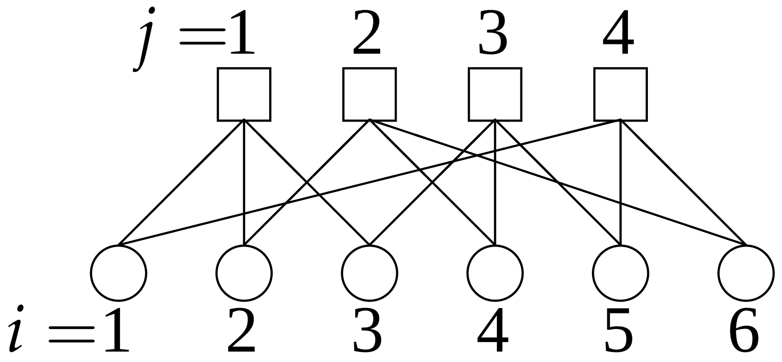
Using

1: for erasure

0: for non-erasure

$$f_{2,0} = x_2$$

The Bit-Oriented Detection

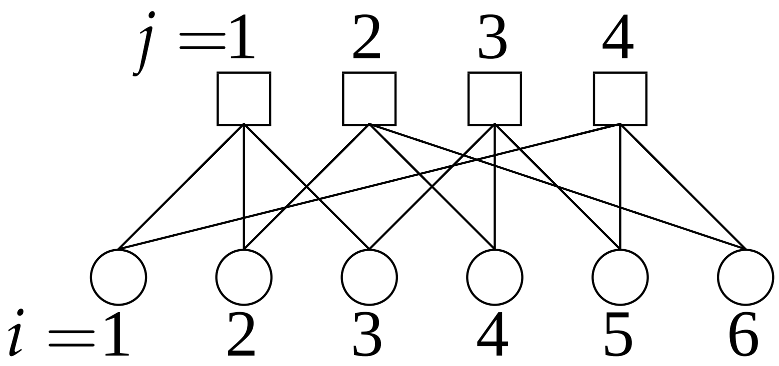


Using
1: for erasure
0: for non-erasure

$$f_{2,0} = x_2$$

$$f_{2,1} = x_2(x_1 + x_3)(x_4 + x_6)$$

The Bit-Oriented Detection



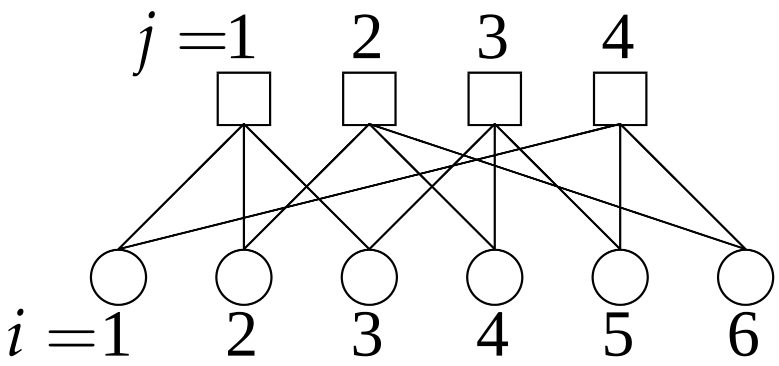
Using

1: for erasure

0: for non-erasure

$$
\begin{aligned}
f_{2,0} &= x_2 \\
f_{2,1} &= x_2(x_1 + x_3)(x_4 + x_6) \\
f_{2,2} &= x_2\left(x_1(f_{5\to 4,1} + f_{6\to 4,1}) + x_3(f_{4\to 3,1} + f_{5\to 3,1})\right) \\
&\quad \cdot \left(x_4(f_{3\to 3,1} + f_{5\to 3,1}) + x_6(f_{1\to 4,1} + f_{5\to 4,1})\right) \\
&= x_2\left(x_1(x_5 + x_6) + x_3(x_4 + x_5)\right) \\
&\quad \cdot \left(x_4(x_3 + x_5) + x_6(x_1 + x_5)\right)
\end{aligned}
$$

The Bit-Oriented Detection



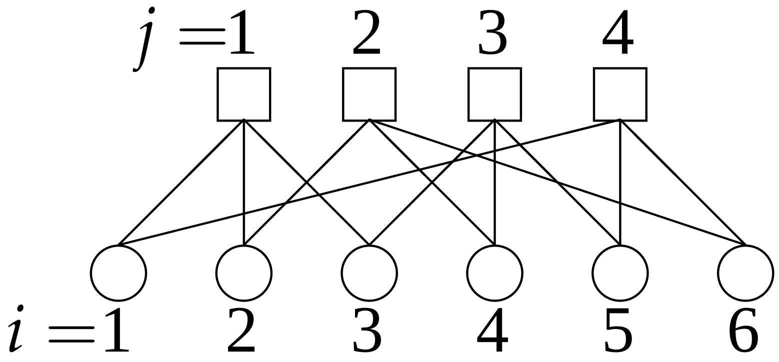
Using

1: for erasure

0: for non-erasure

$$
\begin{aligned}
f_{2,0} &= x_2 \\
f_{2,1} &= x_2(x_1 + x_3)(x_4 + x_6) \\
f_{2,2} &= x_2\left(x_1(f_{5\to4,1} + f_{6\to4,1}) + x_3(f_{4\to3,1} + f_{5\to3,1})\right) \\
&\quad \cdot \left(x_4(f_{3\to3,1} + f_{5\to3,1}) + x_6(f_{1\to4,1} + f_{5\to4,1})\right) \\
&= x_2\left(x_1(x_5 + x_6) + x_3(x_4 + x_5)\right) \\
&\quad \cdot \left(x_4(x_3 + x_5) + x_6(x_1 + x_5)\right) \\
f_2 &:= \lim_{l\to\infty} f_{2,l} = f_{2,2}
\end{aligned}
$$

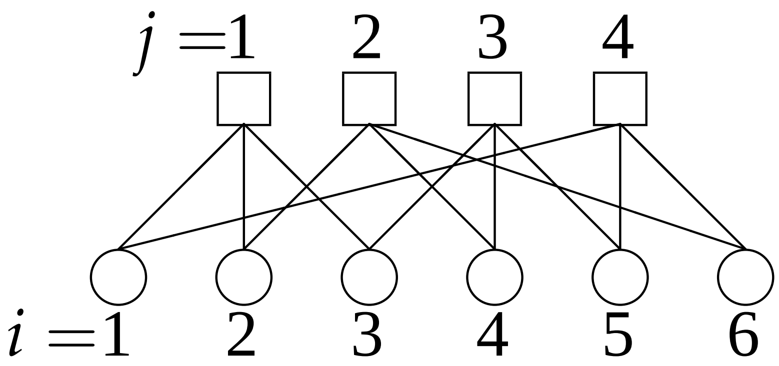
The Bit-Oriented Detection



Using
1: for erasure
0: for non-erasure

$$
\begin{aligned}
f_{2,0} &= x_2 \\
f_{2,1} &= x_2(x_1 + x_3)(x_4 + x_6) \\
f_{2,2} &= x_2\left(x_1(f_{5\to4,1} + f_{6\to4,1}) + x_3(f_{4\to3,1} + f_{5\to3,1})\right) \\
&\quad \cdot \left(x_4(f_{3\to3,1} + f_{5\to3,1}) + x_6(f_{1\to4,1} + f_{5\to4,1})\right) \\
&= x_2\left(x_1(x_5 + x_6) + x_3(x_4 + x_5)\right) \\
&\quad \cdot \left(x_4(x_3 + x_5) + x_6(x_1 + x_5)\right) \\
f_2 &:= \lim_{l\to\infty} f_{2,l} = f_{2,2} \\
p_2 &= \mathsf{E}\{f_2\}
\end{aligned}
$$

The Bit-Oriented Detection



Using
1: for erasure
0: for non-erasure

$$
\begin{aligned}
f_{2,0} &= x_2 \\
f_{2,1} &= \boxed{x_2(x_1 + x_3)(x_4 + x_6)} \\
f_{2,2} &= x_2\left(x_1(f_{5\to4,1} + f_{6\to4,1}) + x_3(f_{4\to3,1} + f_{5\to3,1})\right) \\
&\quad \cdot \left(x_4(f_{3\to3,1} + f_{5\to3,1}) + x_6(f_{1\to4,1} + f_{5\to4,1})\right) \\
&= x_2\left(x_1(x_5 + x_6) + x_3(x_4 + x_5)\right) \\
&\quad \cdot \left(x_4(x_3 + x_5) + x_6(x_1 + x_5)\right) \\
f_2 &:= \lim_{l\to\infty} f_{2,l} = f_{2,2} \\
p_2 &= \mathsf{E}\{f_2\}
\end{aligned}
$$

Asymptotically, no repeated variables

Enumeration-Based LB & UB

$$\begin{aligned} f_2 &= x_2\left(x_1(x_5+x_6)+x_3(x_4+x_5)\right)\left(x_4(x_3+x_5)+x_6(x_1+x_5)\right) \\ &= x_1x_2x_6+x_2x_3x_4+x_1x_2x_4x_5+x_2x_3x_5x_6 \end{aligned}$$

Each product term corresponds to an irreducible stopping set.

Enumeration-Based LB & UB

$$
\begin{aligned}
f_2 &= x_2\left(x_1(x_5+x_6)+x_3(x_4+x_5)\right)\left(x_4(x_3+x_5)+x_6(x_1+x_5)\right) \\
&= x_1x_2x_6 + x_2x_3x_4 + x_1x_2x_4x_5 + x_2x_3x_5x_6
\end{aligned}
$$

Each product term corresponds to an irreducible stopping set.

- LB: Graph-based search [Richardson 03], iteration-based relaxation [Yedidia *et al.* 01]

$$
\begin{aligned}
f_{2,\mathrm{LB}_a} &= x_2x_3x_4 & \quad f_{2,\mathrm{LB}_b} &= x_1x_2x_4x_5 \\
\mathrm{LB}_a &= \mathsf{E}\{f_{2,\mathrm{LB}}\} = \epsilon^3 & \quad \mathrm{LB}_b &= \mathsf{E}\{f_{2,\mathrm{LB}_b}\} = \epsilon^4
\end{aligned}
$$

Enumeration-Based LB & UB

$$\begin{aligned} f_2 &= x_2\left(x_1(x_5+x_6)+x_3(x_4+x_5)\right)\left(x_4(x_3+x_5)+x_6(x_1+x_5)\right) \\ &= x_1x_2x_6+x_2x_3x_4+x_1x_2x_4x_5+x_2x_3x_5x_6 \end{aligned}$$

Each product term corresponds to an irreducible stopping set.

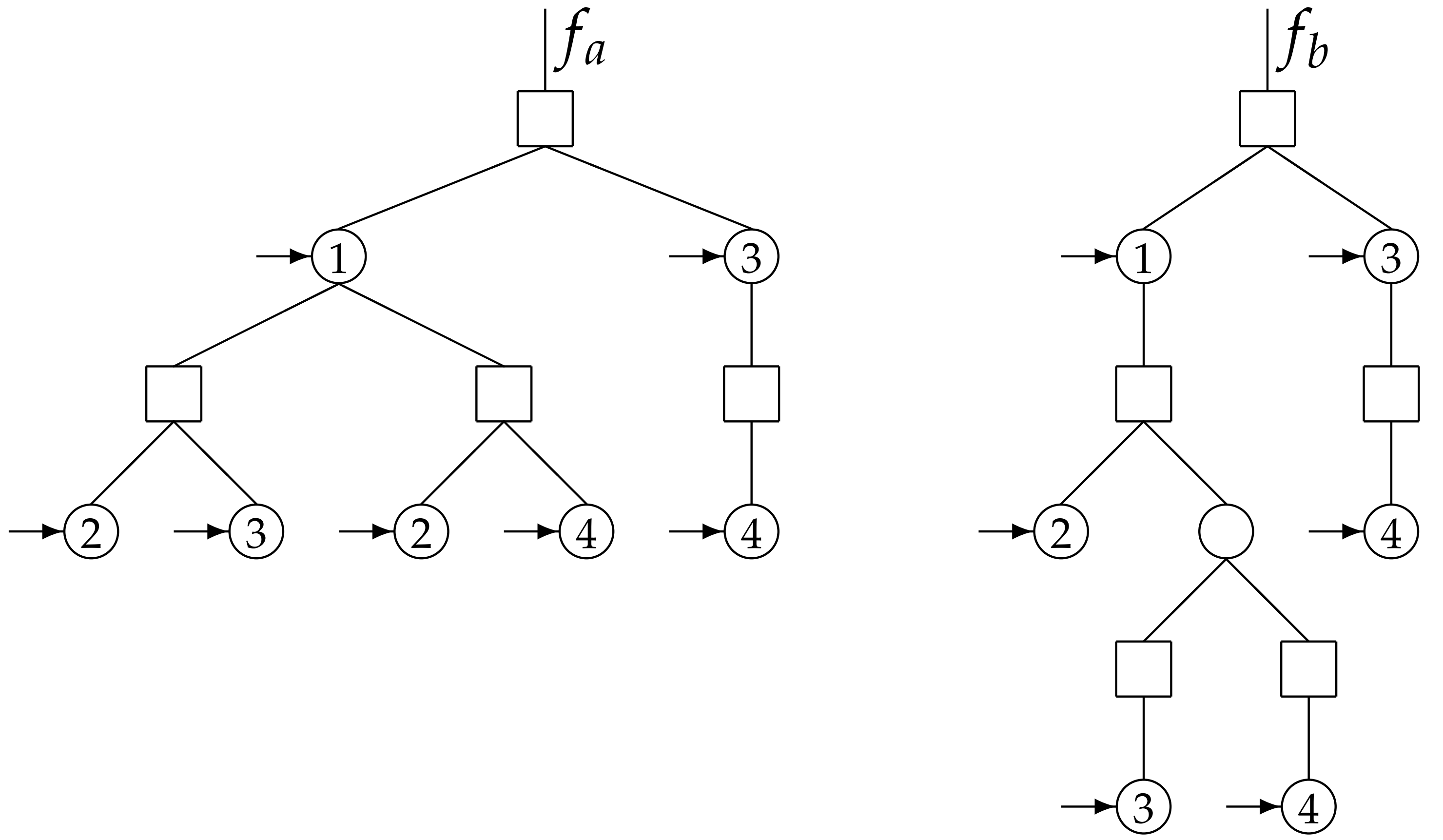
- LB: Graph-based search [Richardson 03], iteration-based relaxation [Yedidia *et al.* 01]

$$\begin{aligned} f_{2,\mathrm{LB}_a} &= x_2x_3x_4 & f_{2,\mathrm{LB}_b} &= x_1x_2x_4x_5 \\ \mathrm{LB}_a &= \mathsf{E}\{f_{2,\mathrm{LB}}\} = \epsilon^3 & \mathrm{LB}_b &= \mathsf{E}\{f_{2,\mathrm{LB}_b}\} = \epsilon^4 \end{aligned}$$

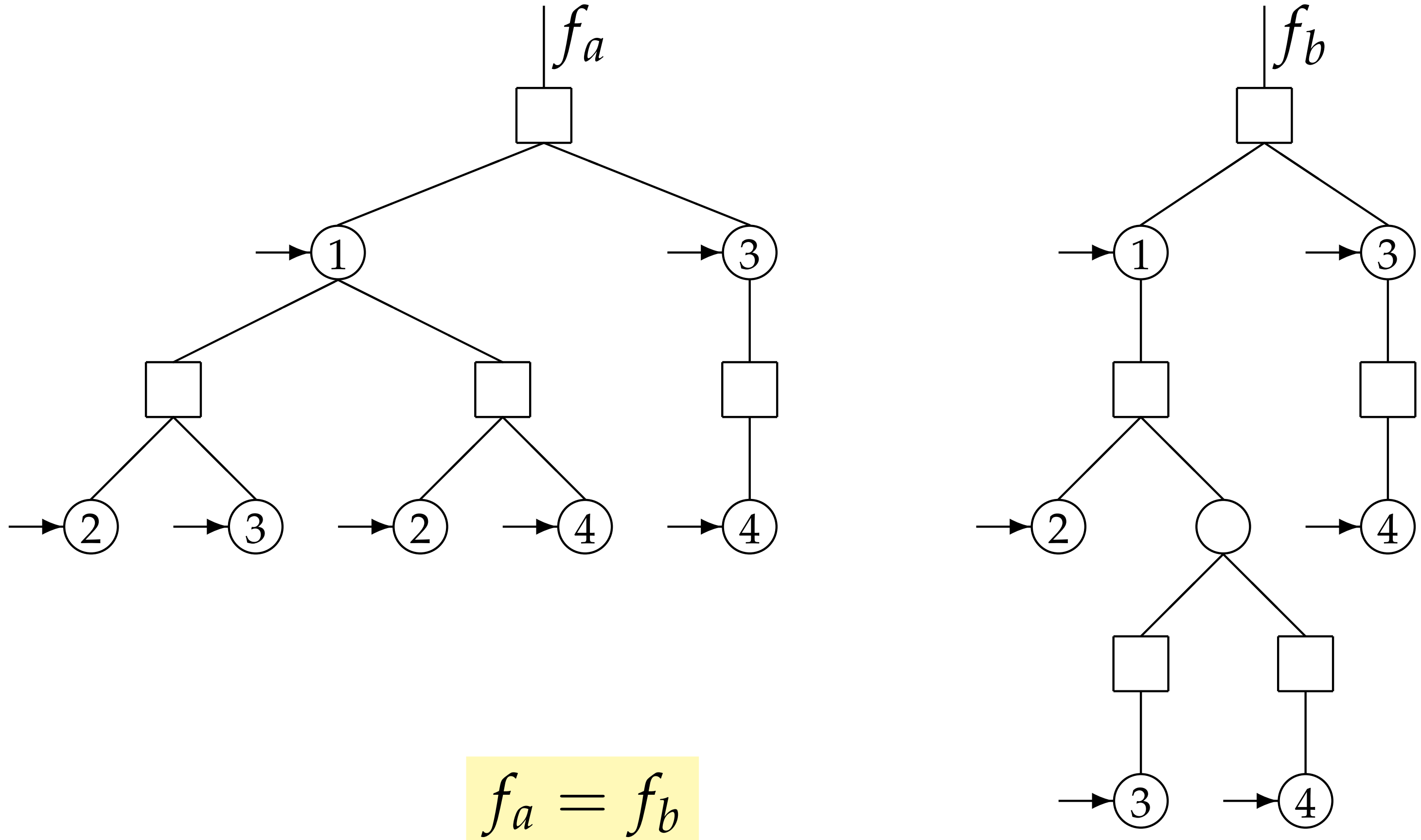
- UB: Iteration-based relaxation [Yedidia *et al.* 01]

$$\begin{aligned} f_{2,2} &= x_2x_3x_4+x_1x_2x_4x_5+x_1x_2x_6+x_2x_3x_5x_6 \\ &\leq x_2 1 x_4 + 1 x_2 x_4 1 + 1 x_2 x_6 + x_2 1\cdot 1 x_6 = x_2x_4+x_2x_6 \\ \mathrm{UB} &= 2\epsilon^2-\epsilon^3 \end{aligned}$$

An Upper Bound



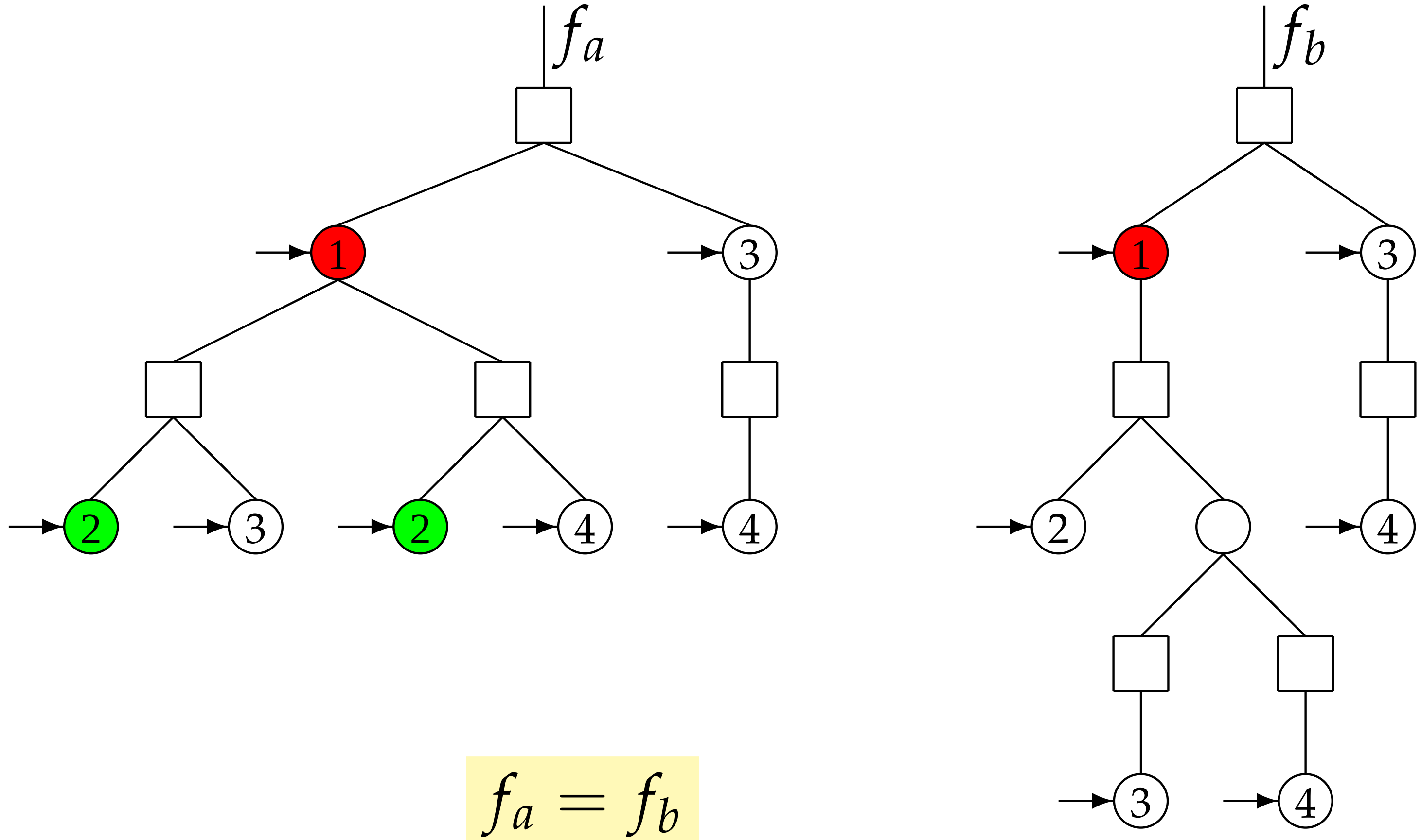
An Upper Bound



$$f_a = f_b$$

$$\mathcal{E}_{DE}\{f_a\} < p_e = \mathsf{E}\{f_a\} = \mathsf{E}\{f_b\} \leq \mathcal{E}_{DE}\{f_b\}$$

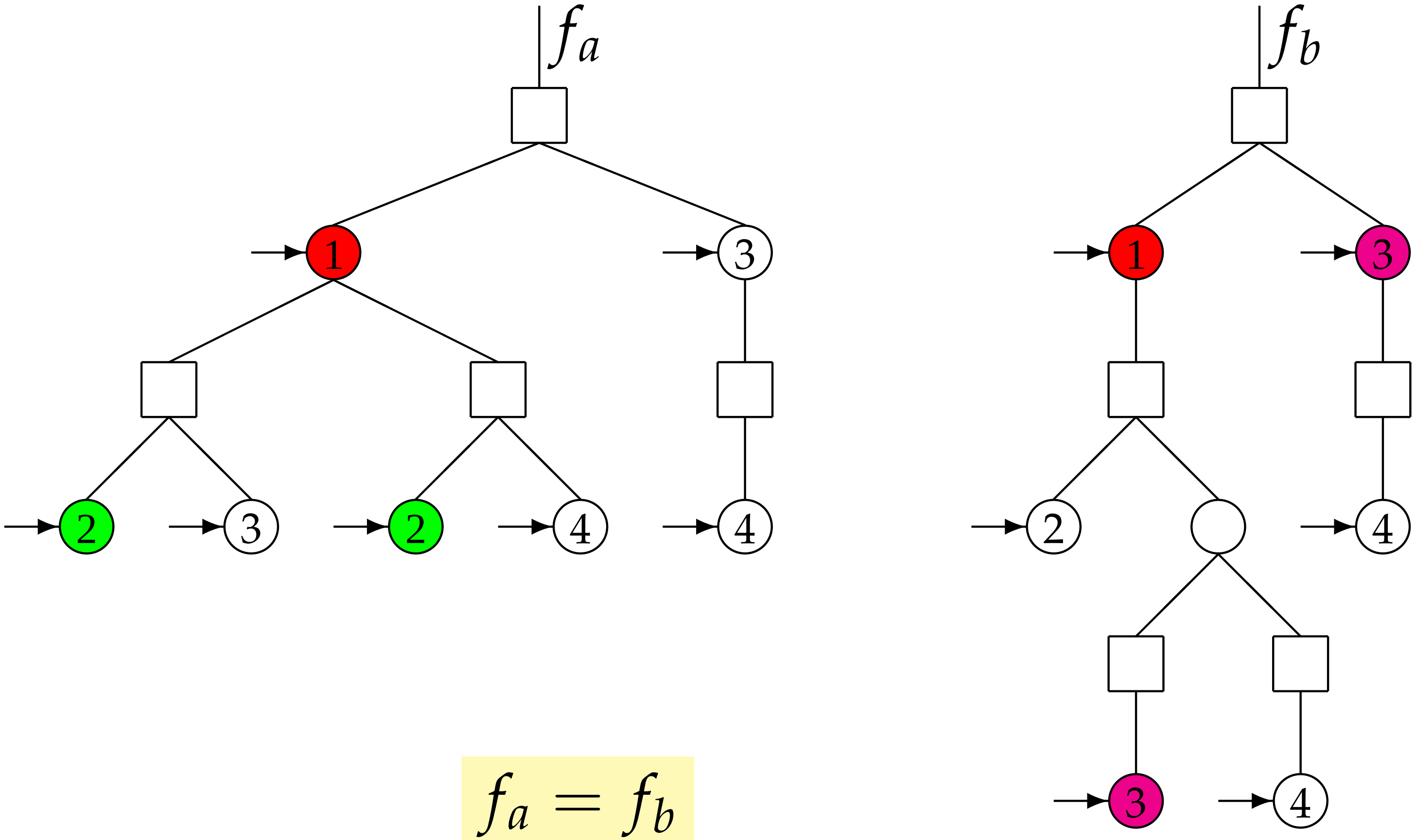
An Upper Bound



$$f_a = f_b$$

$$\mathcal{E}_{DE}\{f_a\} < p_e = \mathsf{E}\{f_a\} = \mathsf{E}\{f_b\} \leq \mathcal{E}_{DE}\{f_b\}$$

An Upper Bound



$$f_a = f_b$$

$$\mathcal{E}_{DE}\{f_a\} < p_e = \mathsf{E}\{f_a\} = \mathsf{E}\{f_b\} \leq \mathcal{E}_{DE}\{f_b\}$$

An Upper Bound (Cont'd)

Theorem 1 *Suppose all messages entering any variable node are independent, or equivalently, the youngest common ancestors of all pairs of repeated bits are check nodes.*

$$\mathcal{E}_{DE}\{f\} \geq \mathsf{E}\{f\}$$

Theorem 2 *This upper bound is tight in order.*

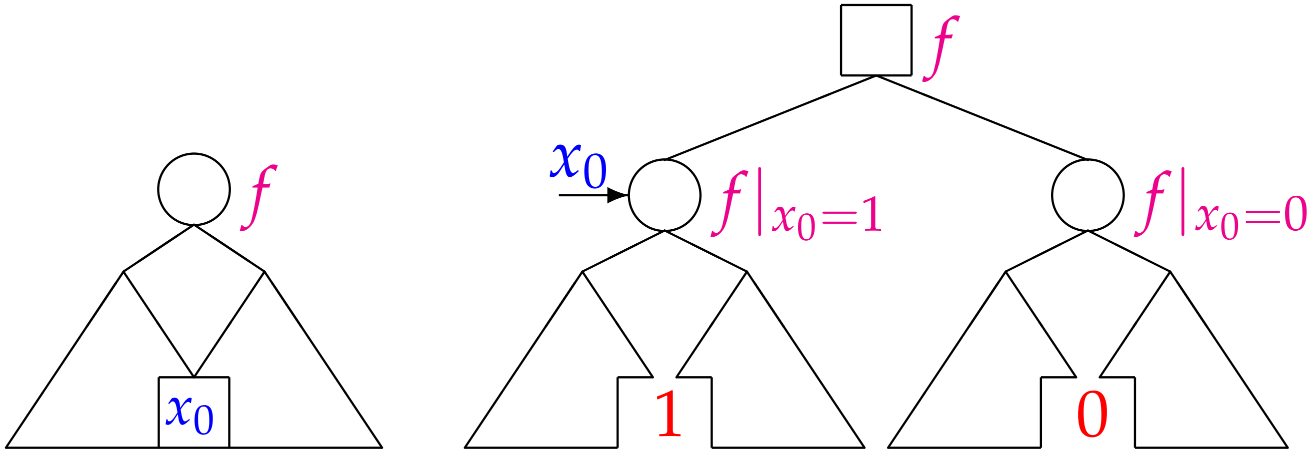
$$\mathcal{E}_{DE}\{f\}(\epsilon) = O(\mathsf{E}\{f\}(\epsilon)), \ \forall \epsilon$$

A Pivoting Rule

$$f = x_0 \cdot f|_{x_0=1} + f|_{x_0=0}$$

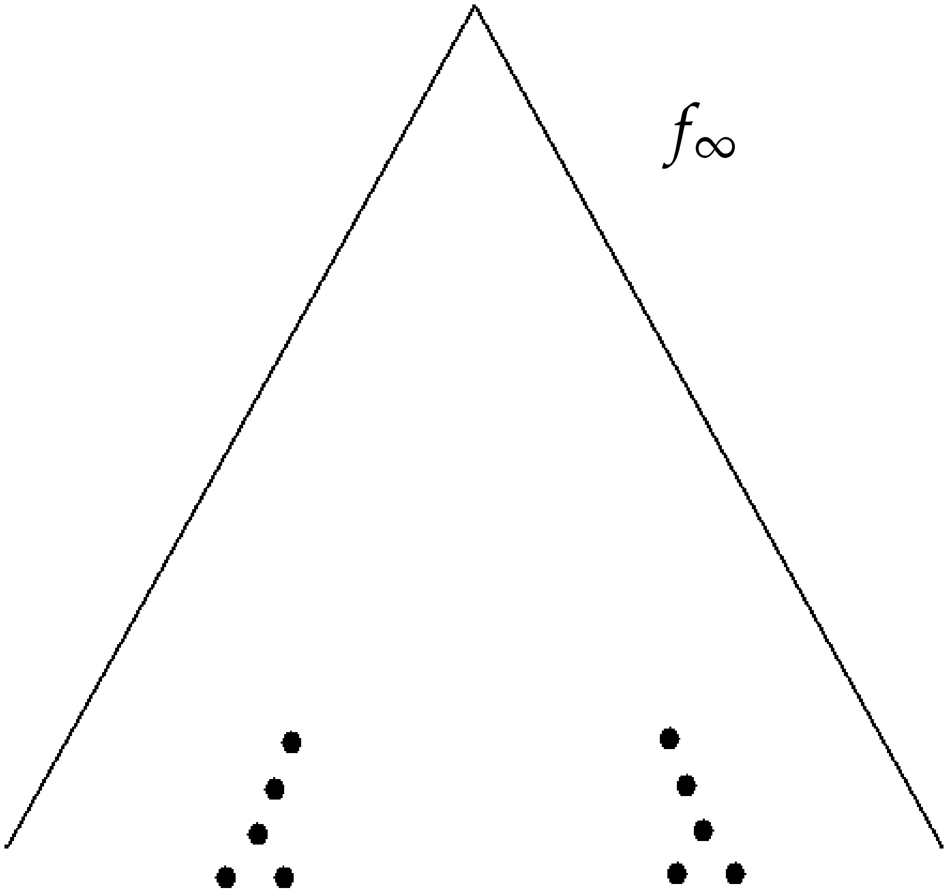
A Pivoting Rule

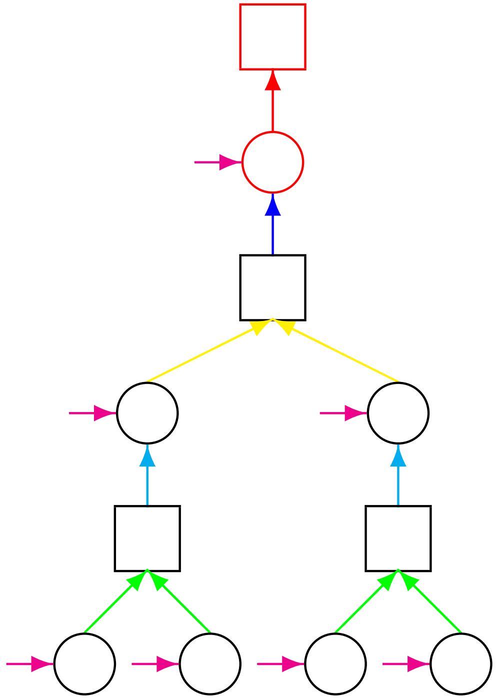
$$f = x_0 \cdot f|_{x_0=1} + f|_{x_0=0}$$



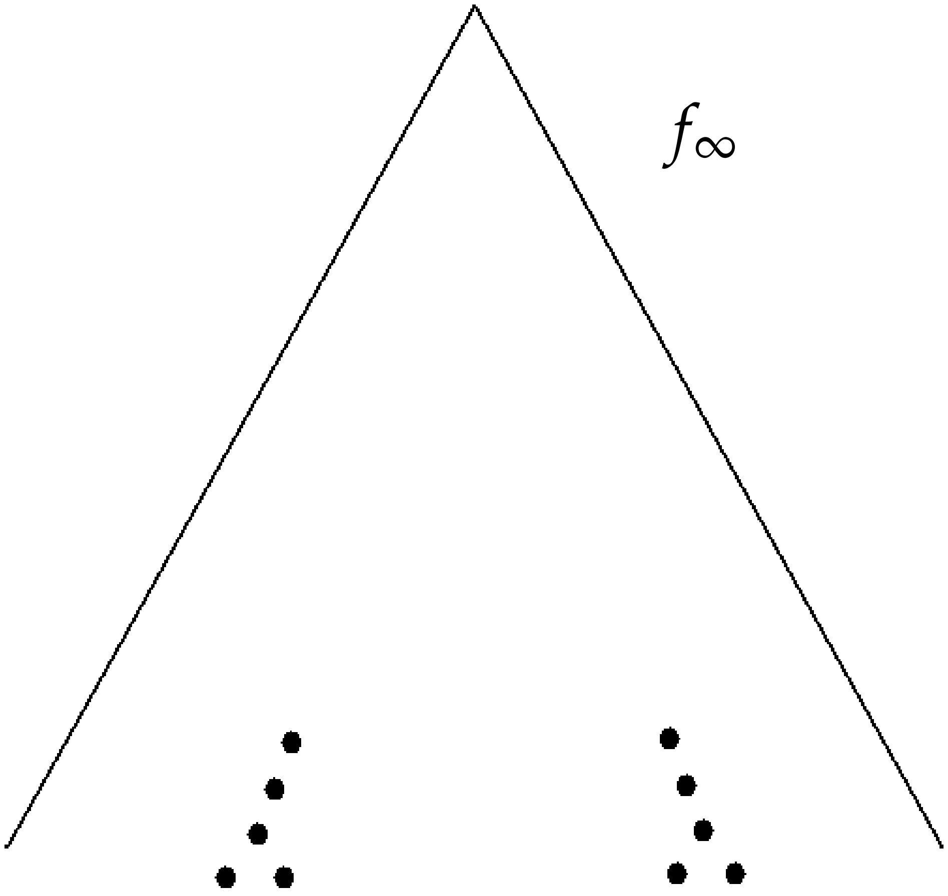
(a) Original (b) Decoupled

The Two-Stage Algorithm

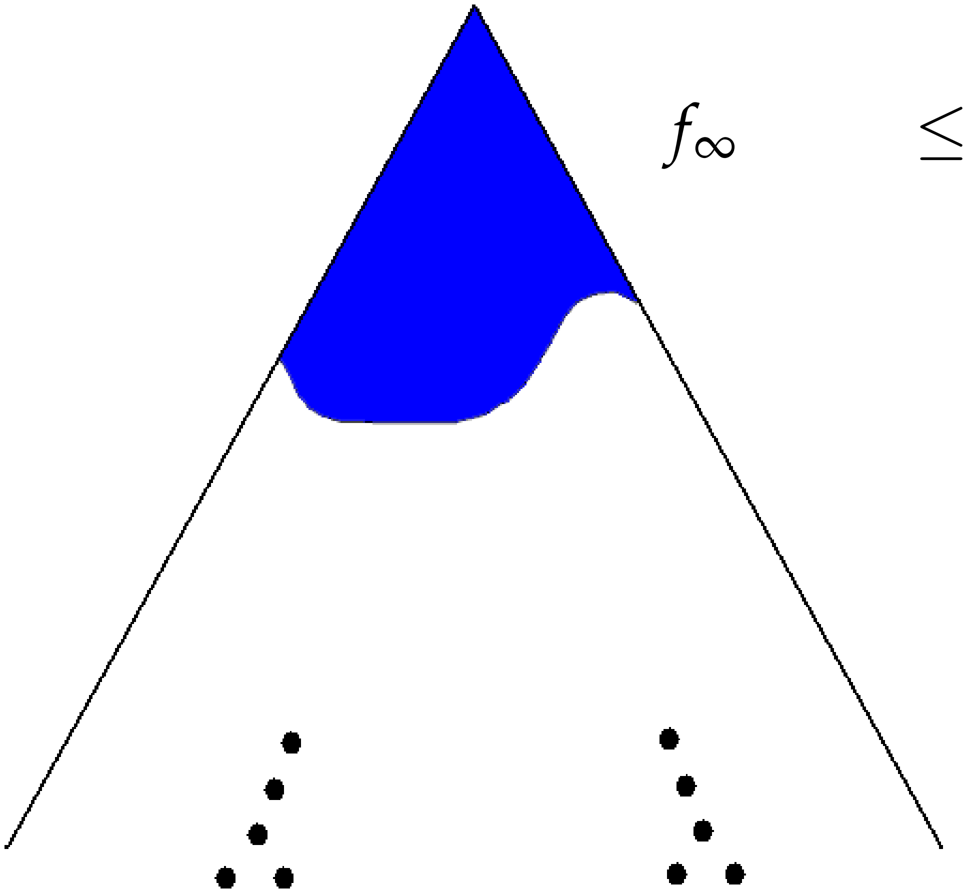




The Two-Stage Algorithm

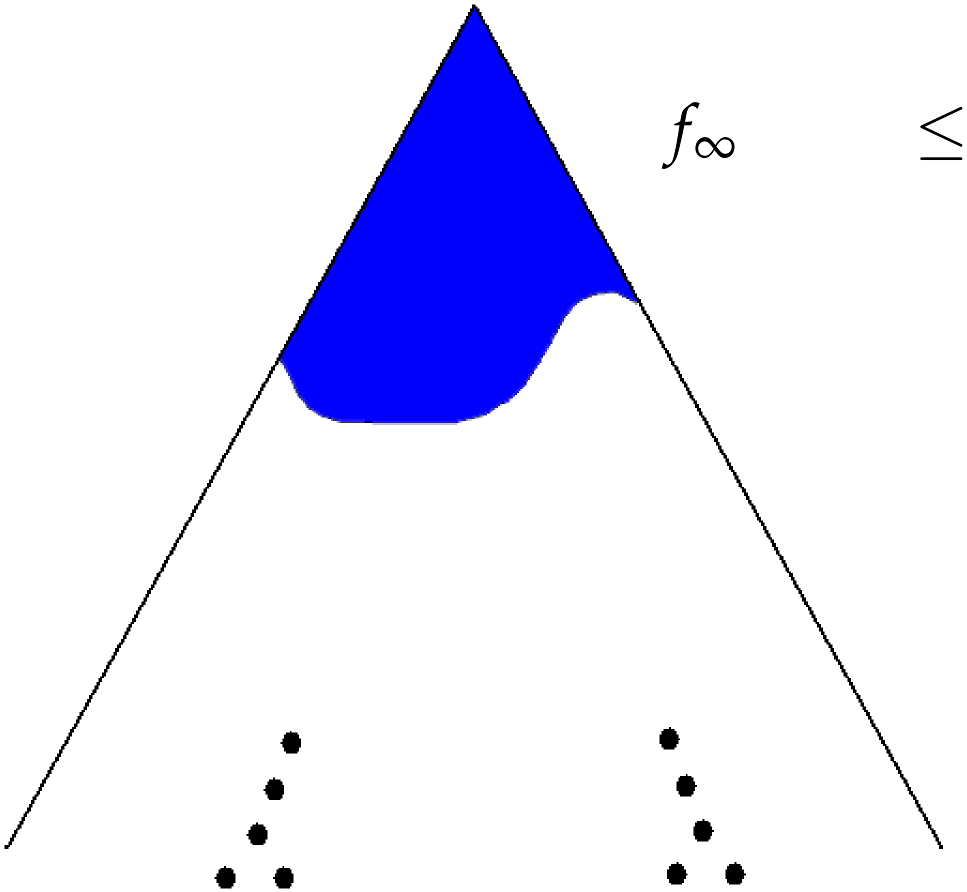


The Two-Stage Algorithm

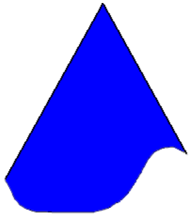


$$f_\infty \leq f_{FINITE}$$

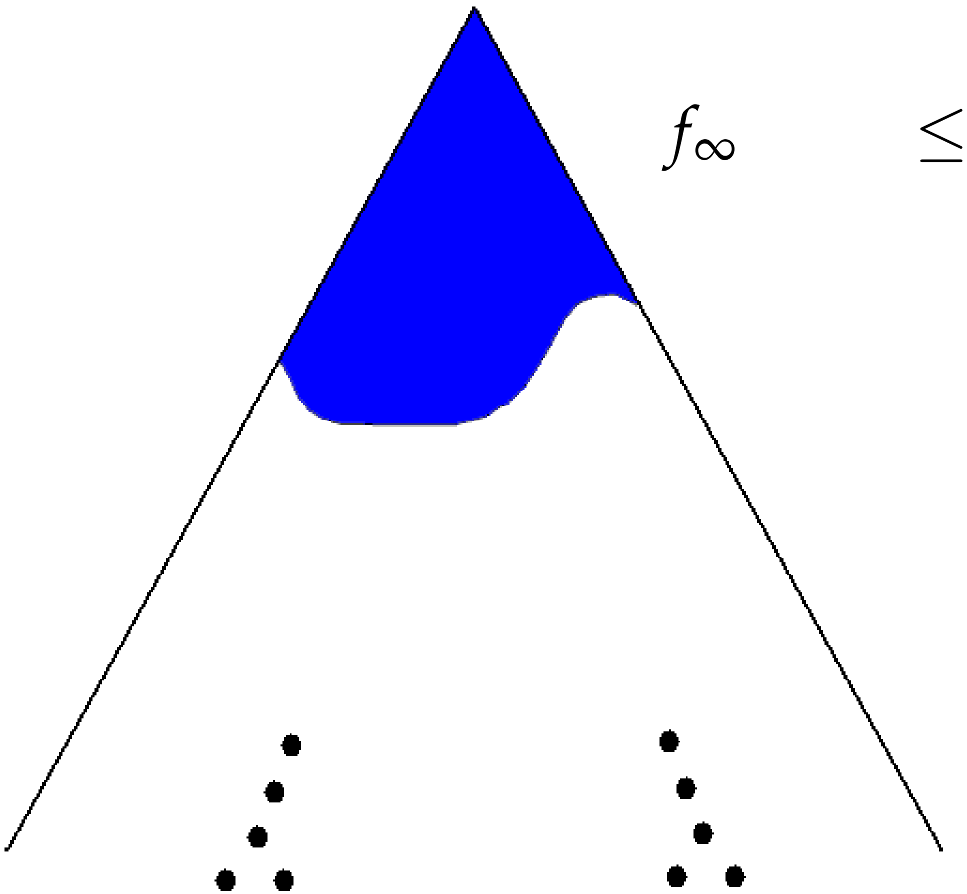
The Two-Stage Algorithm



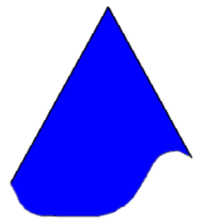
$f_\infty \quad \leq \quad f_{FINITE}$

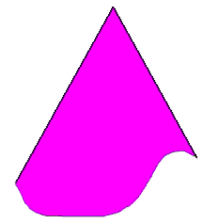


The Two-Stage Algorithm



$$f_{\infty} \leq f_{FINITE} = \tilde{f}_{FINITE}$$



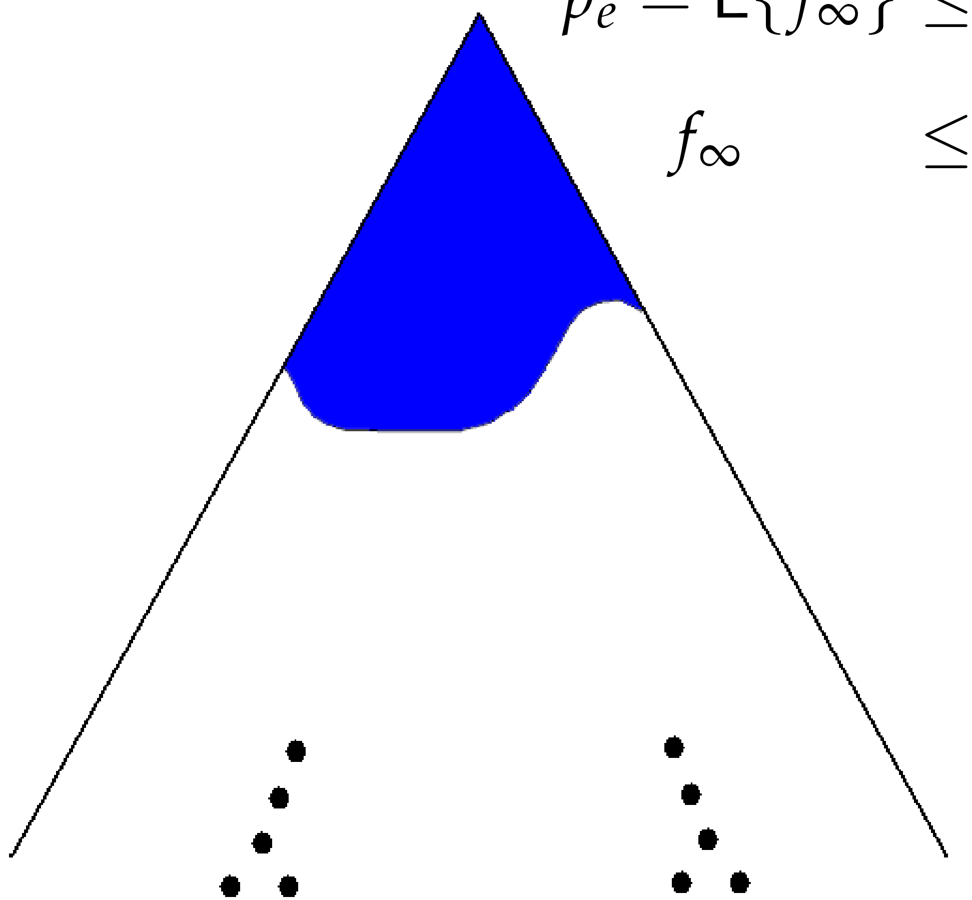


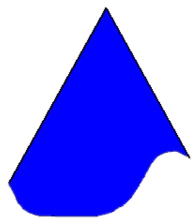
The Two-Stage Algorithm

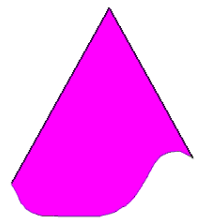
w. the same order

$$p_e = \mathsf{E}\{f_\infty\} \leq \mathsf{E}\{f_{FINITE}\} \leq \mathcal{E}_{DE}\{\tilde{f}_{FINITE}\}$$

$$f_\infty \leq f_{FINITE} = \tilde{f}_{FINITE}$$





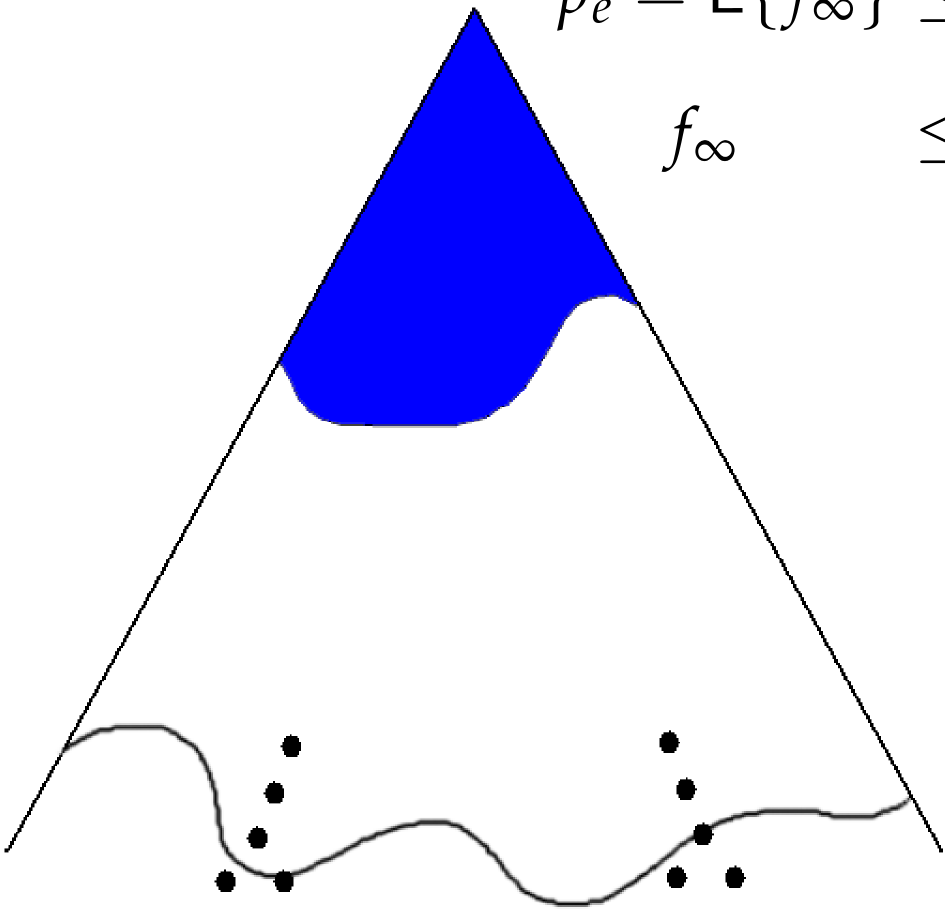


The Two-Stage Algorithm

w. the same order

$$p_e = \mathsf{E}\{f_\infty\} \leq \mathsf{E}\{f_{FINITE}\} \leq \mathcal{E}_{DE}\{\tilde{f}_{FINITE}\}$$

$$f_\infty \leq f_{FINITE} = \tilde{f}_{FINITE}$$

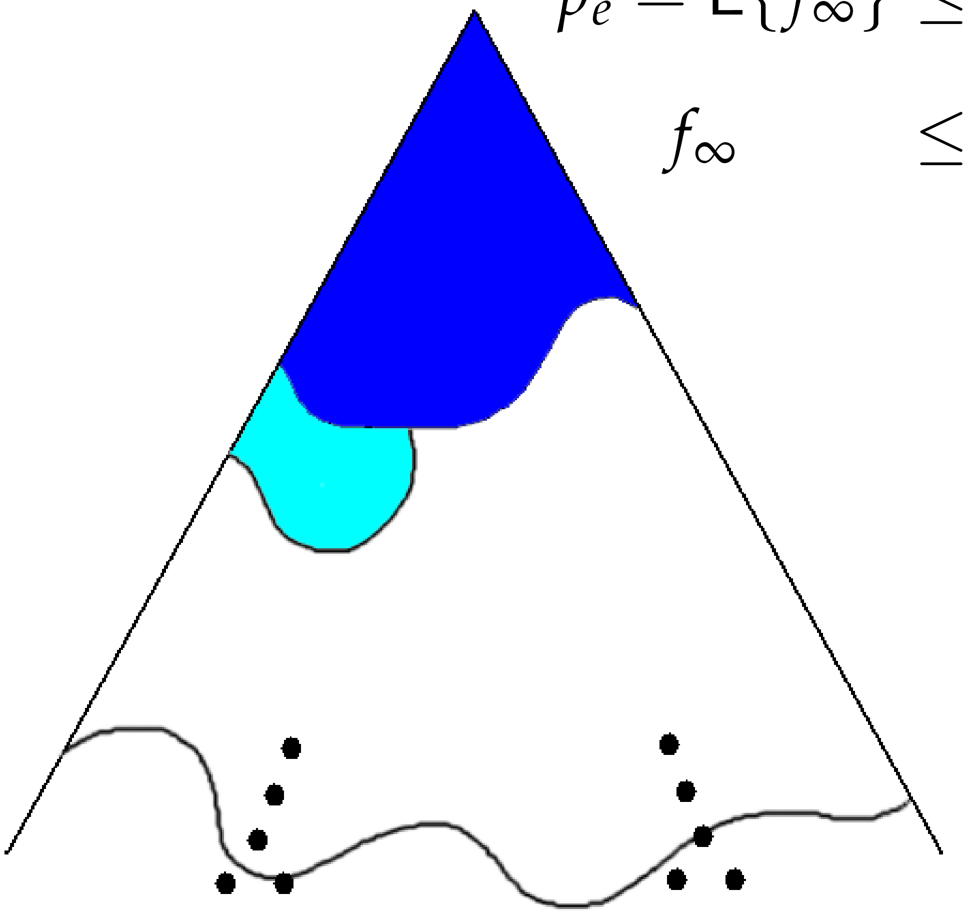


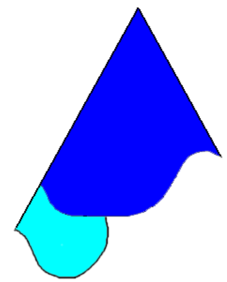
The Two-Stage Algorithm

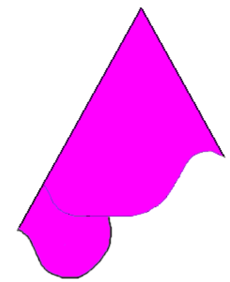
w. the same order

$$p_e = \mathsf{E}\{f_\infty\} \le \mathsf{E}\{f_{FINITE}\} \le \mathcal{E}_{DE}\{\tilde{f}_{FINITE}\}$$

$$f_\infty \le f_{FINITE} = \tilde{f}_{FINITE}$$





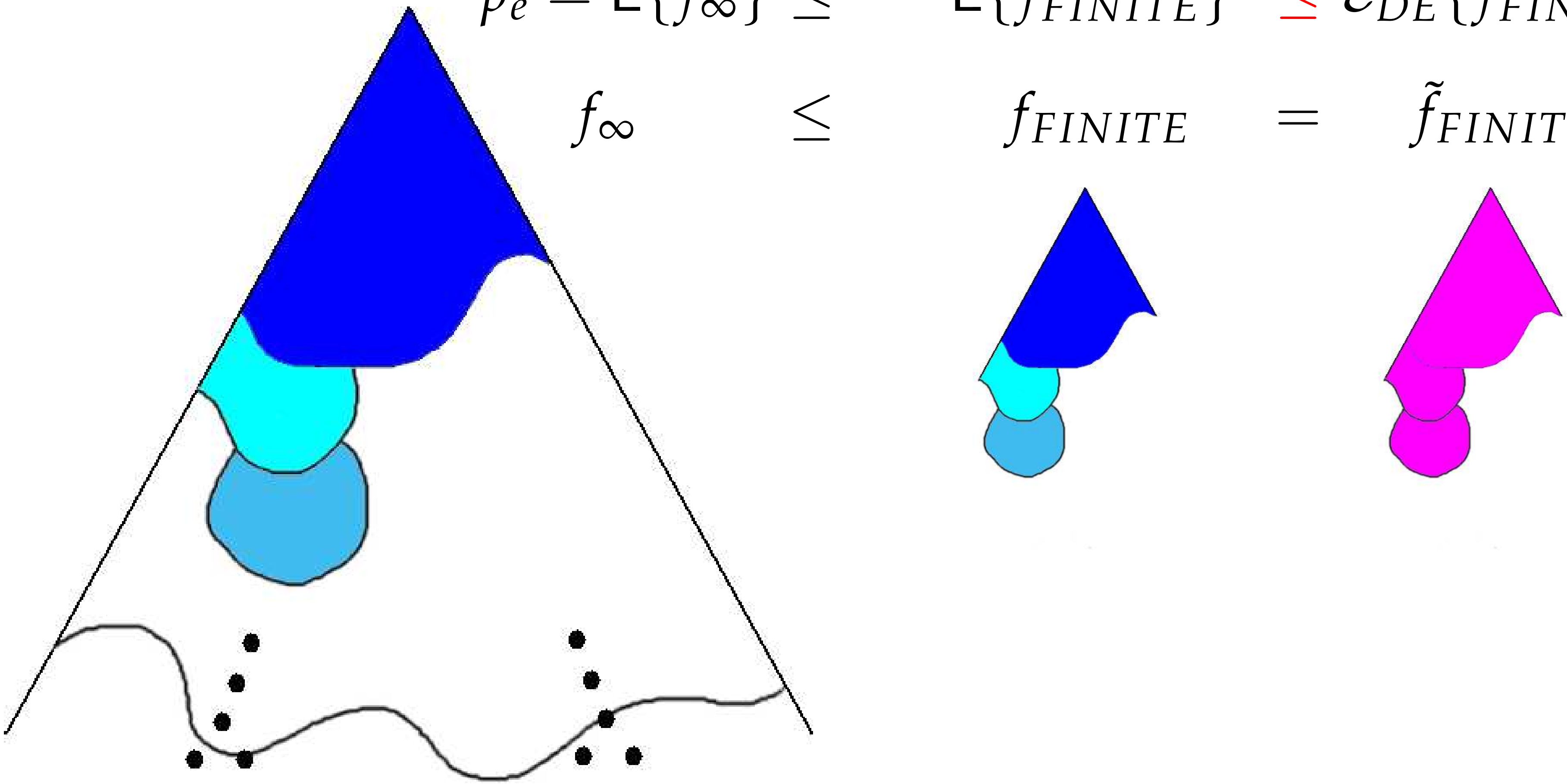


The Two-Stage Algorithm

w. the same order

$$p_e = \mathsf{E}\{f_\infty\} \leq \mathsf{E}\{f_{FINITE}\} \leq \mathcal{E}_{DE}\{\tilde{f}_{FINITE}\}$$

$$f_\infty \leq f_{FINITE} = \tilde{f}_{FINITE}$$

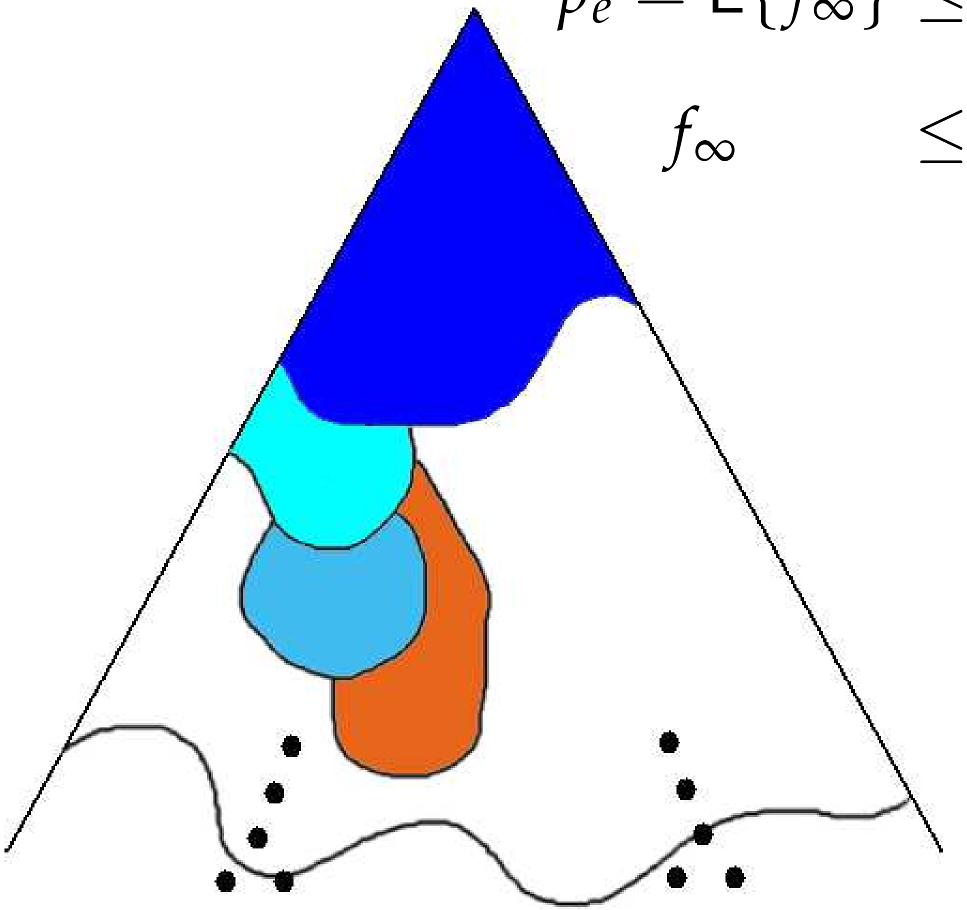


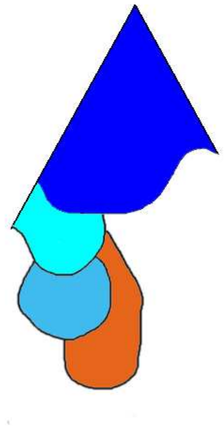
The Two-Stage Algorithm

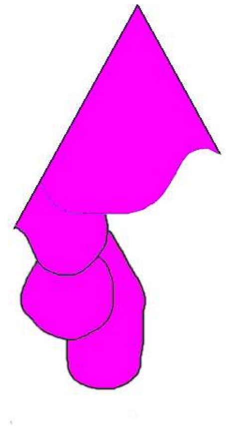
w. the same order

$$p_e = \mathsf{E}\{f_\infty\} \leq \quad \mathsf{E}\{f_{FINITE}\} \leq \mathcal{E}_{DE}\{\tilde{f}_{FINITE}\}$$

$$f_\infty \quad \leq \quad f_{FINITE} \quad = \quad \tilde{f}_{FINITE}$$





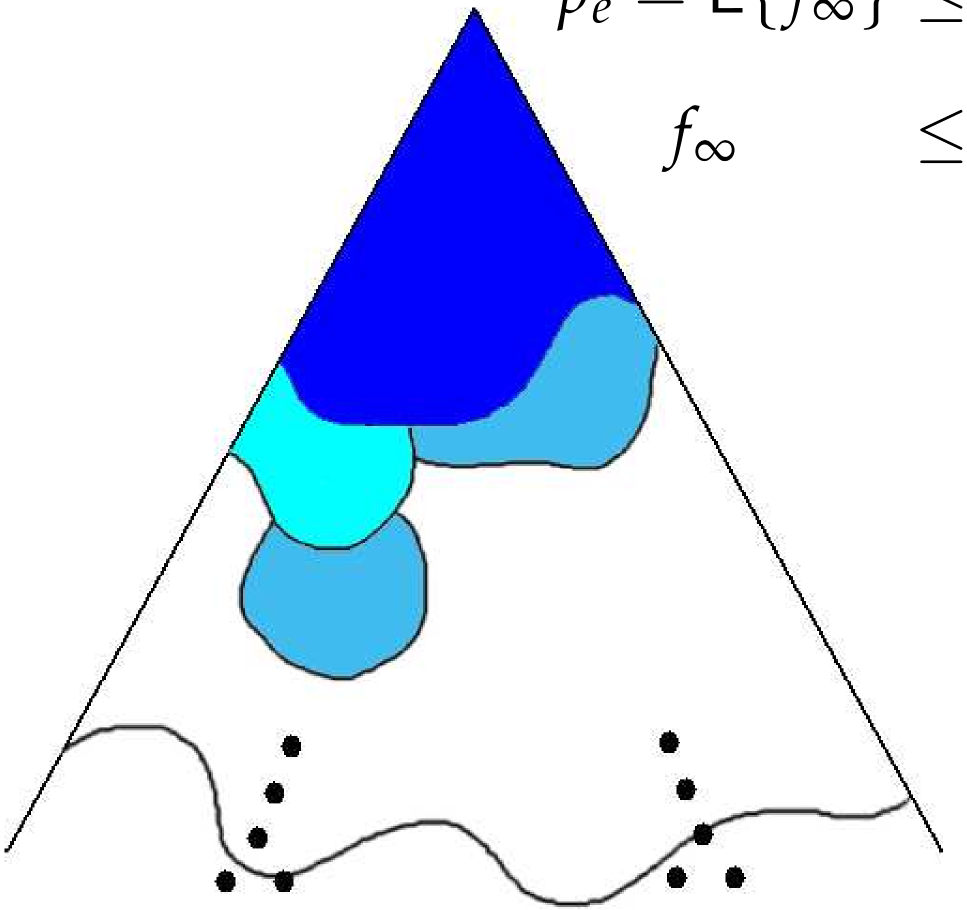


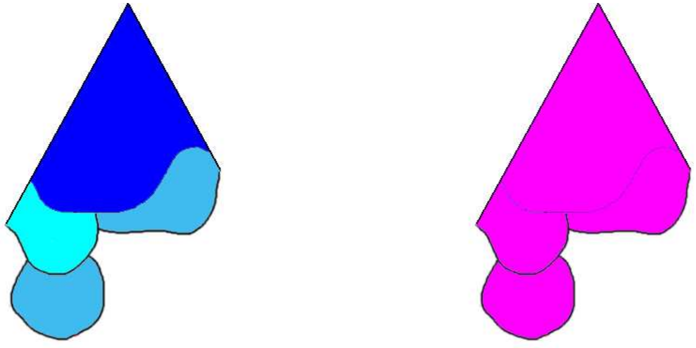
The Two-Stage Algorithm

w. the same order

$$p_e = \mathsf{E}\{f_\infty\} \leq \mathsf{E}\{f_{FINITE}\} \leq \mathcal{E}_{DE}\{\tilde{f}_{FINITE}\}$$

$$f_\infty \leq f_{FINITE} = \tilde{f}_{FINITE}$$



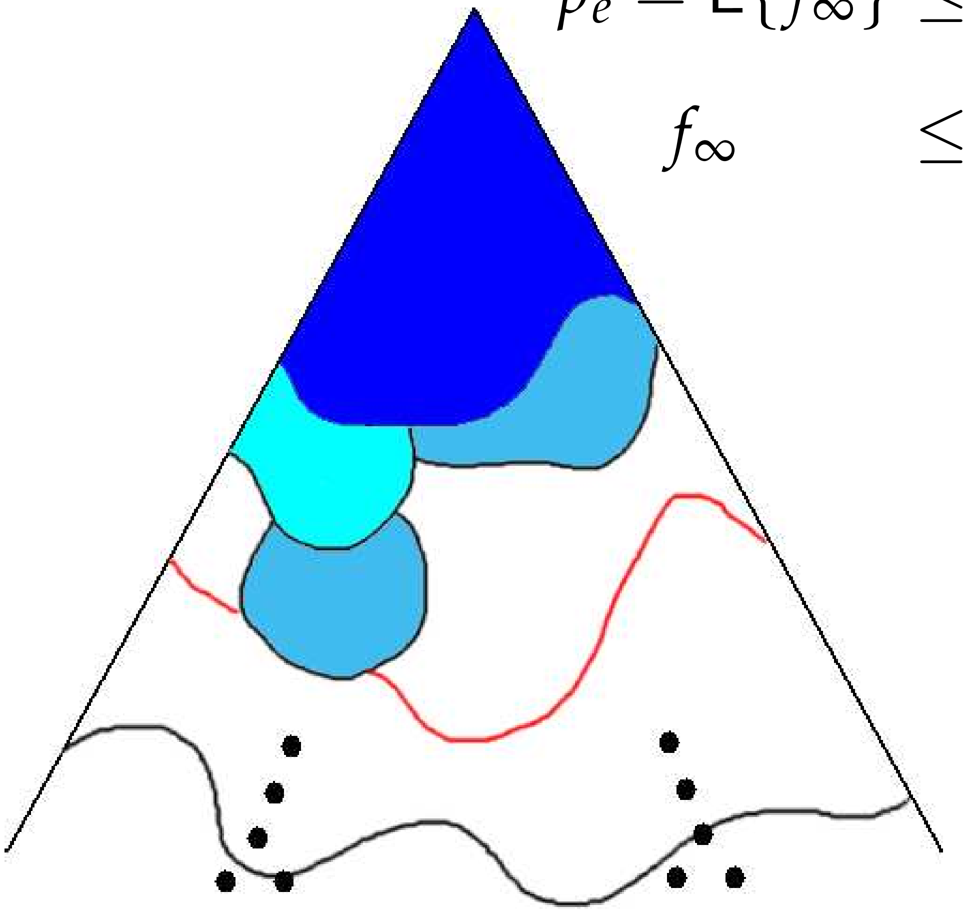


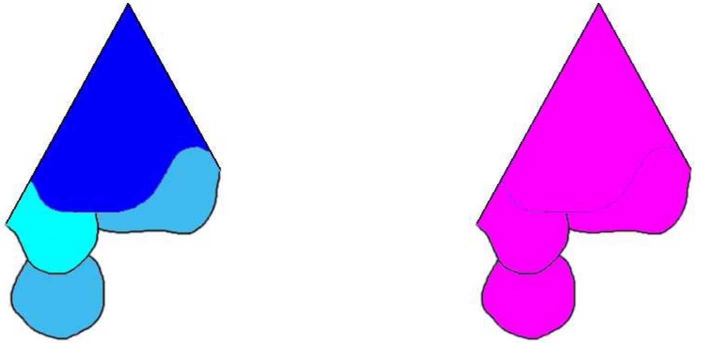
The Two-Stage Algorithm

w. the same order

$$p_e = \mathsf{E}\{f_\infty\} \leq \mathsf{E}\{f_{FINITE}\} \leq \mathcal{E}_{DE}\{\tilde{f}_{FINITE}\}$$

$$f_\infty \leq f_{FINITE} = \tilde{f}_{FINITE}$$



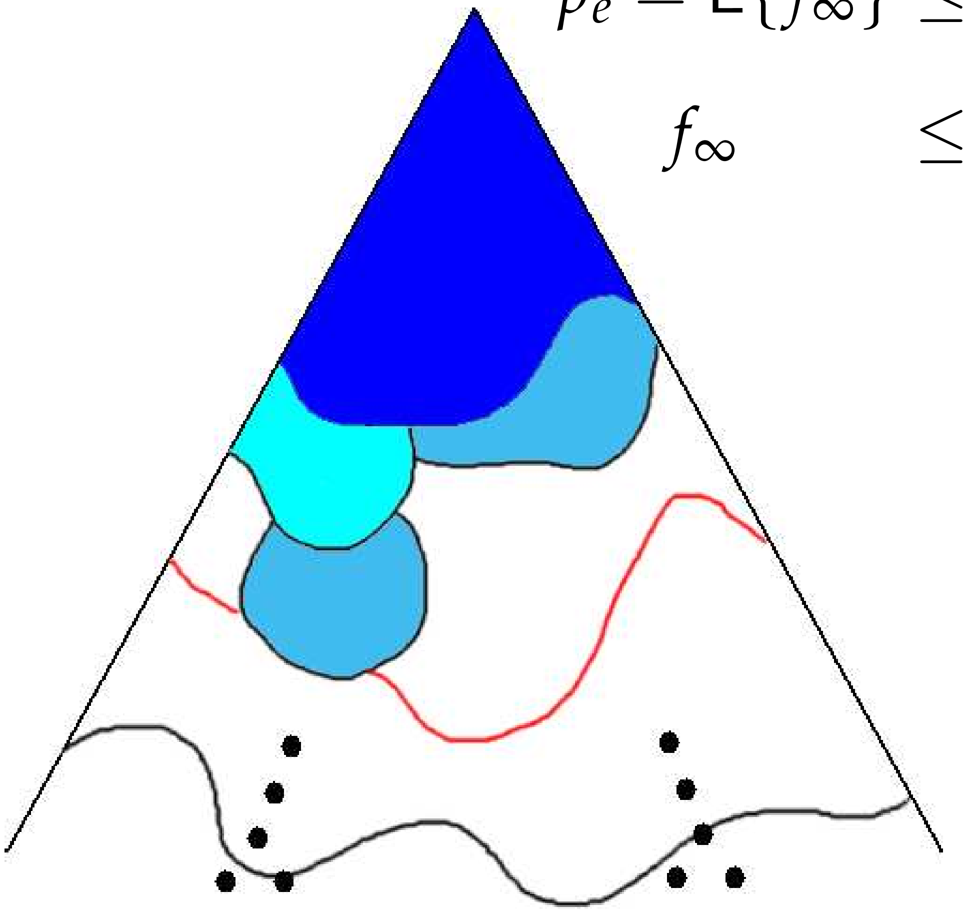


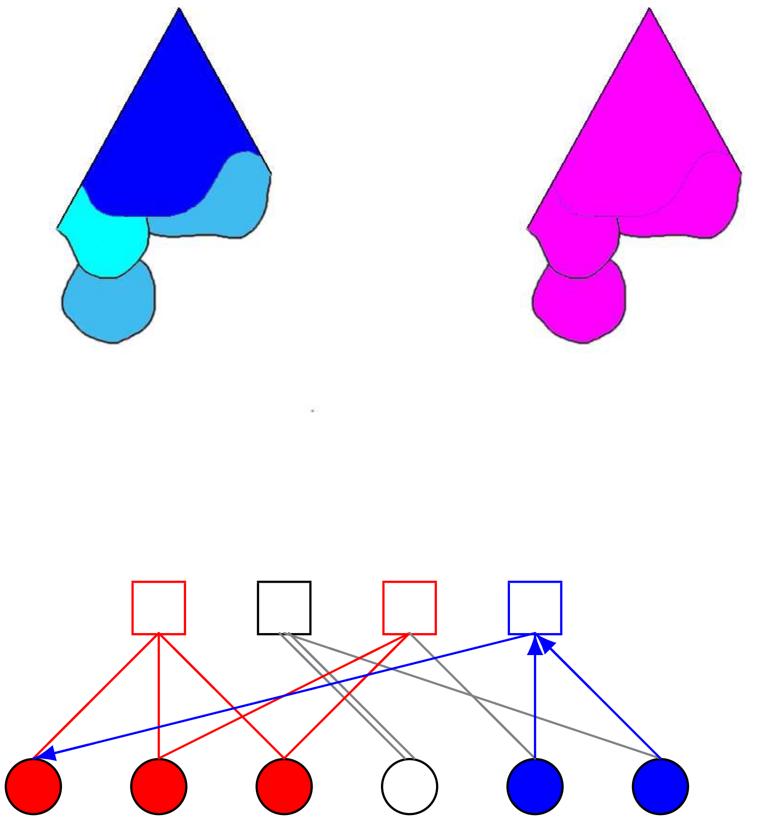
The Two-Stage Algorithm

w. the same order

$$p_e = \mathsf{E}\{f_\infty\} \leq \mathsf{E}\{f_{FINITE}\} \leq \mathcal{E}_{DE}\{\tilde{f}_{FINITE}\}$$

$$f_\infty \leq f_{FINITE} = \tilde{f}_{FINITE}$$





A Narrowing Search

Theorem 3 (Asymptotically Tight) *With an "optimal" growing module, we have*

$$\mathrm{UB}(\epsilon) = \mathcal{O}(p_e(\epsilon)).$$

A Narrowing Search

Theorem 3 (Asymptotically Tight) *With an "optimal" growing module, we have*

$$\mathrm{UB}(\epsilon) = \mathcal{O}(p_e(\epsilon)).$$

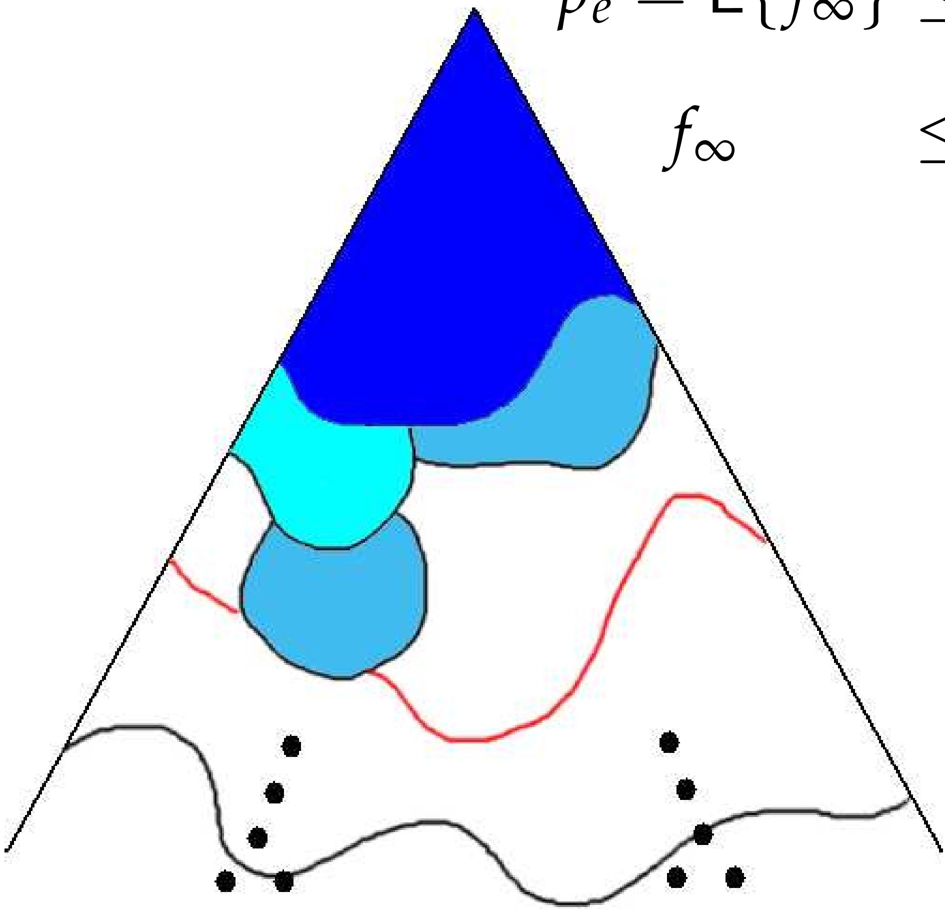
Theorem 4 (Exhaustive Enumeration)

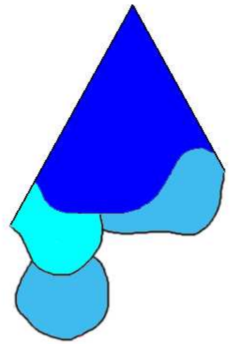
- $d := \min\{\text{size}(\mathbf{x}) | \forall \mathbf{x}$ *such that* $\tilde{f}_{FINITE}(\mathbf{x}) = 1\}$. *Then the stopping distance* $\geq d$.
- *If there exists such a minimum* **x** *being a SS, then ALL minimum SSs are in the set of all minimum* **x**.
- *The exhaustive list of minimum SSs leads to a lower bound tight in both order and multiplicity.*

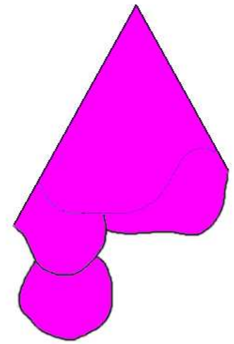
The Two-Stage Algorithm

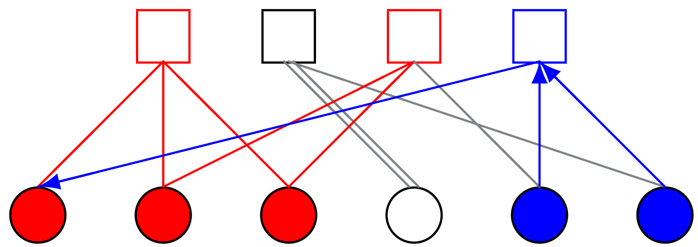
$$p_e = \mathrm{E}\{f_\infty\} \leq \mathrm{E}\{f_{FINITE}\} \leq \mathcal{E}_{DE}\{\tilde{f}_{FINITE}\}$$

$$f_\infty \leq f_{FINITE} = \tilde{f}_{FINITE}$$









Numerical Experiments

- The (7,4,3) Hamming code:

$$\mathbf{H} = \begin{pmatrix} 0 & 0 & 1 & 1 & 1 & 0 & 1 \\ 0 & 1 & 0 & 1 & 0 & 1 & 1 \\ 1 & 0 & 0 & 0 & 1 & 1 & 1 \end{pmatrix}$$

Numerical Experiments

- The (7,4,3) Hamming code:

$$\mathbf{H} = \begin{pmatrix} 0 & 0 & 1 & 1 & 1 & 0 & 1 \\ 0 & 1 & 0 & 1 & 0 & 1 & 1 \\ 1 & 0 & 0 & 0 & 1 & 1 & 1 \end{pmatrix}$$

$$\forall i \in [1,7], \forall \epsilon \in (0,1], \quad \frac{\mathrm{UB}_i(\epsilon)}{p_i(\epsilon)} \leq 1.3.$$

Numerical Experiments

- The (7,4,3) Hamming code:

$$\mathbf{H} = \begin{pmatrix} 0 & 0 & 1 & 1 & 1 & 0 & 1 \\ 0 & 1 & 0 & 1 & 0 & 1 & 1 \\ 1 & 0 & 0 & 0 & 1 & 1 & 1 \end{pmatrix}$$

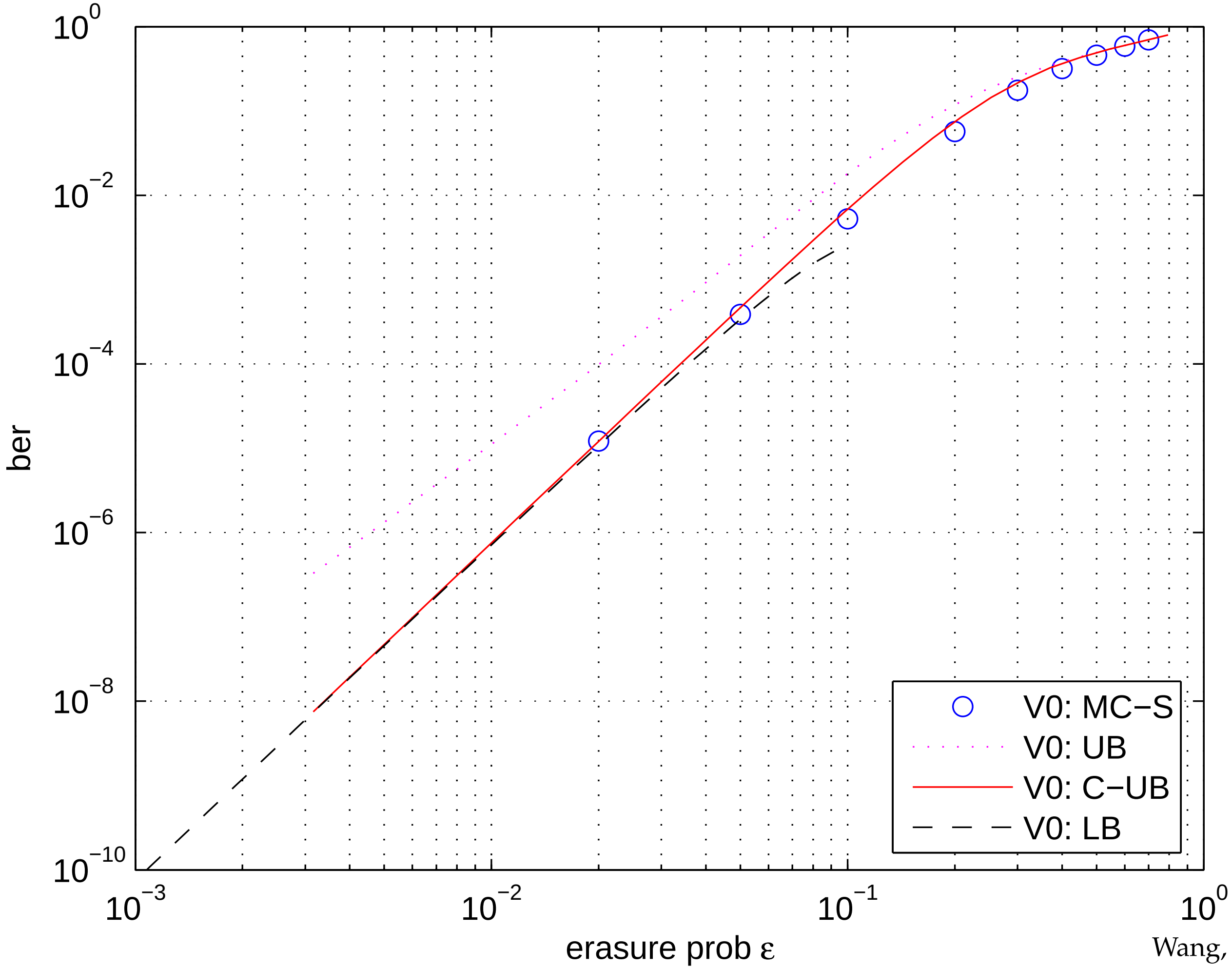
$$\forall i \in [1,7], \forall \epsilon \in (0,1], \quad \frac{\mathrm{UB}_i(\epsilon)}{p_i(\epsilon)} \leq 1.3.$$

- The (23,12,7) Golay code:

$$\mathbf{H} = [\mathbf{H}'\mathbf{I}] \text{ in which } \mathbf{H}' = \begin{pmatrix} 1 & 0 & 0 & 1 & 1 & 1 & 0 & 0 & 0 & 1 & 1 & 1 \\ 1 & 0 & 1 & 0 & 1 & 1 & 0 & 1 & 1 & 0 & 0 & 1 \\ 1 & 0 & 1 & 1 & 0 & 1 & 1 & 0 & 1 & 0 & 1 & 0 \\ 1 & 0 & 1 & 1 & 1 & 0 & 1 & 1 & 0 & 1 & 0 & 0 \\ 1 & 1 & 0 & 0 & 1 & 1 & 1 & 0 & 1 & 1 & 0 & 0 \\ 1 & 1 & 0 & 1 & 0 & 1 & 1 & 1 & 0 & 0 & 0 & 1 \\ 1 & 1 & 0 & 1 & 1 & 0 & 0 & 1 & 1 & 0 & 1 & 0 \\ 1 & 1 & 1 & 0 & 0 & 1 & 0 & 1 & 0 & 1 & 1 & 0 \\ 1 & 1 & 1 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 1 & 1 \\ 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 1 \\ 0 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \end{pmatrix}$$

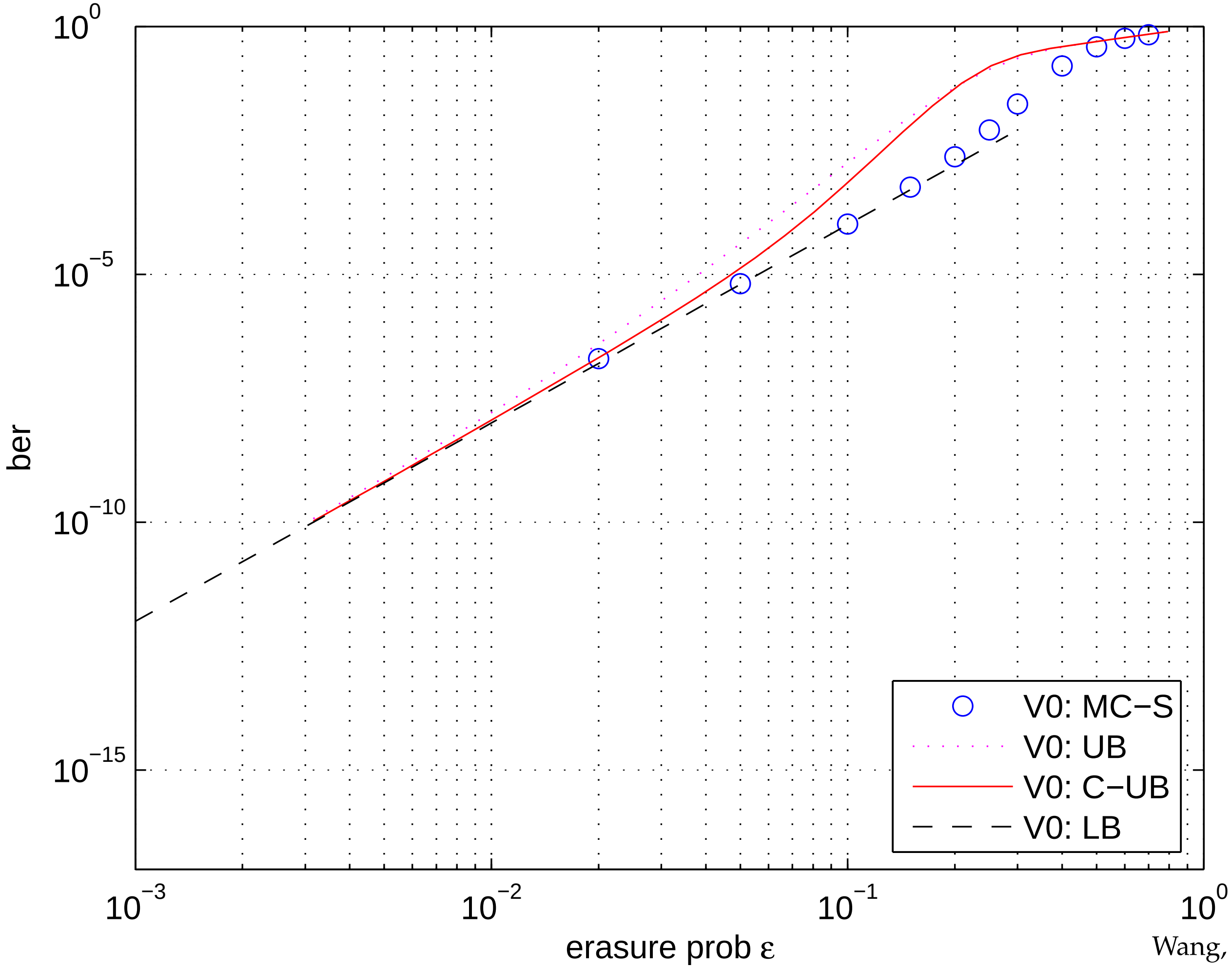
Numerical Experiments (cont'd)

- The (23,12,7) Golay code:
  - bit 0, (4*,75*)



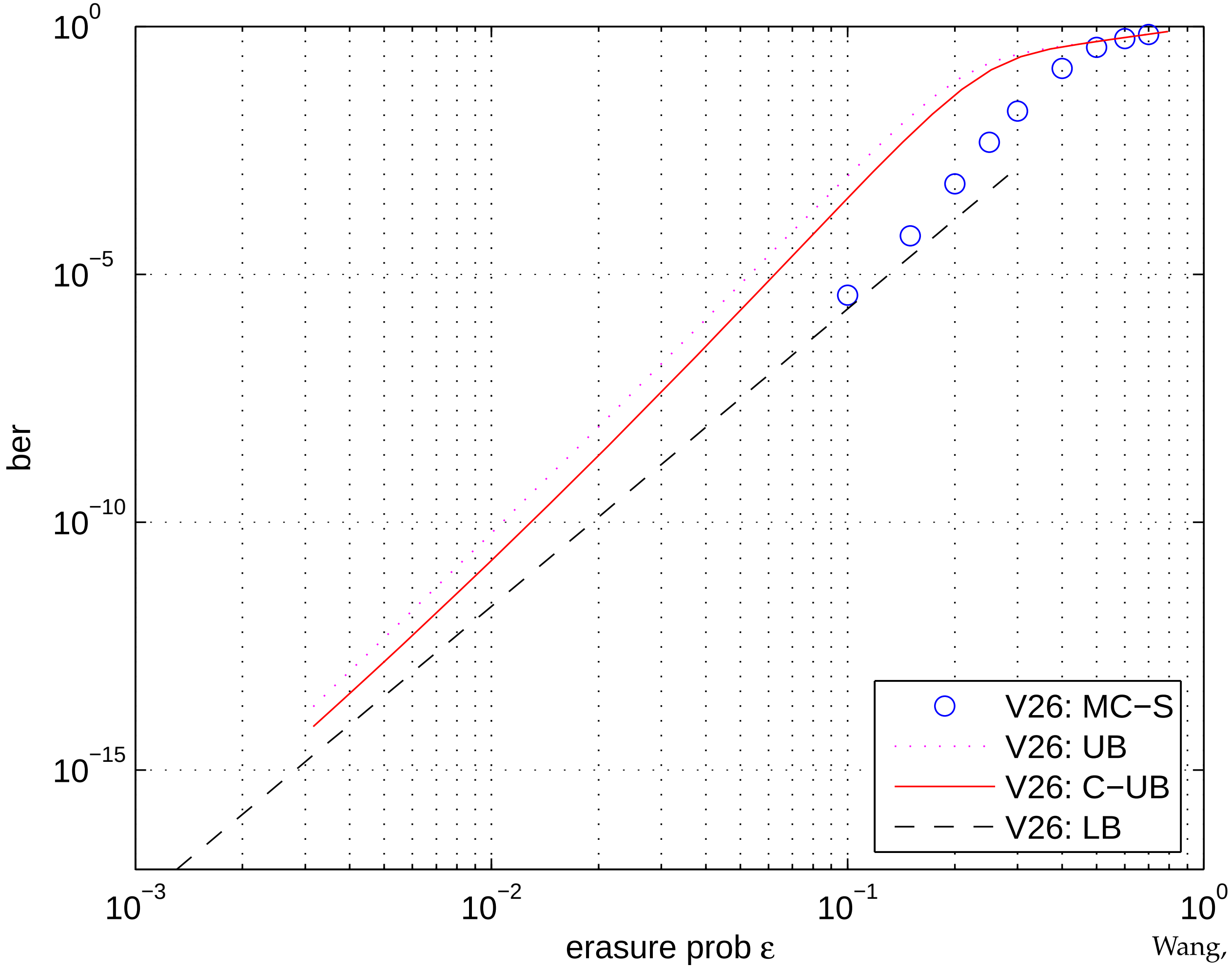
Numerical Experiments (cont’d)

- A fixed, finite LDPC code with var deg. 3, chk deg. 6, $n = 50$:
  - bit 0, (4*,1*)



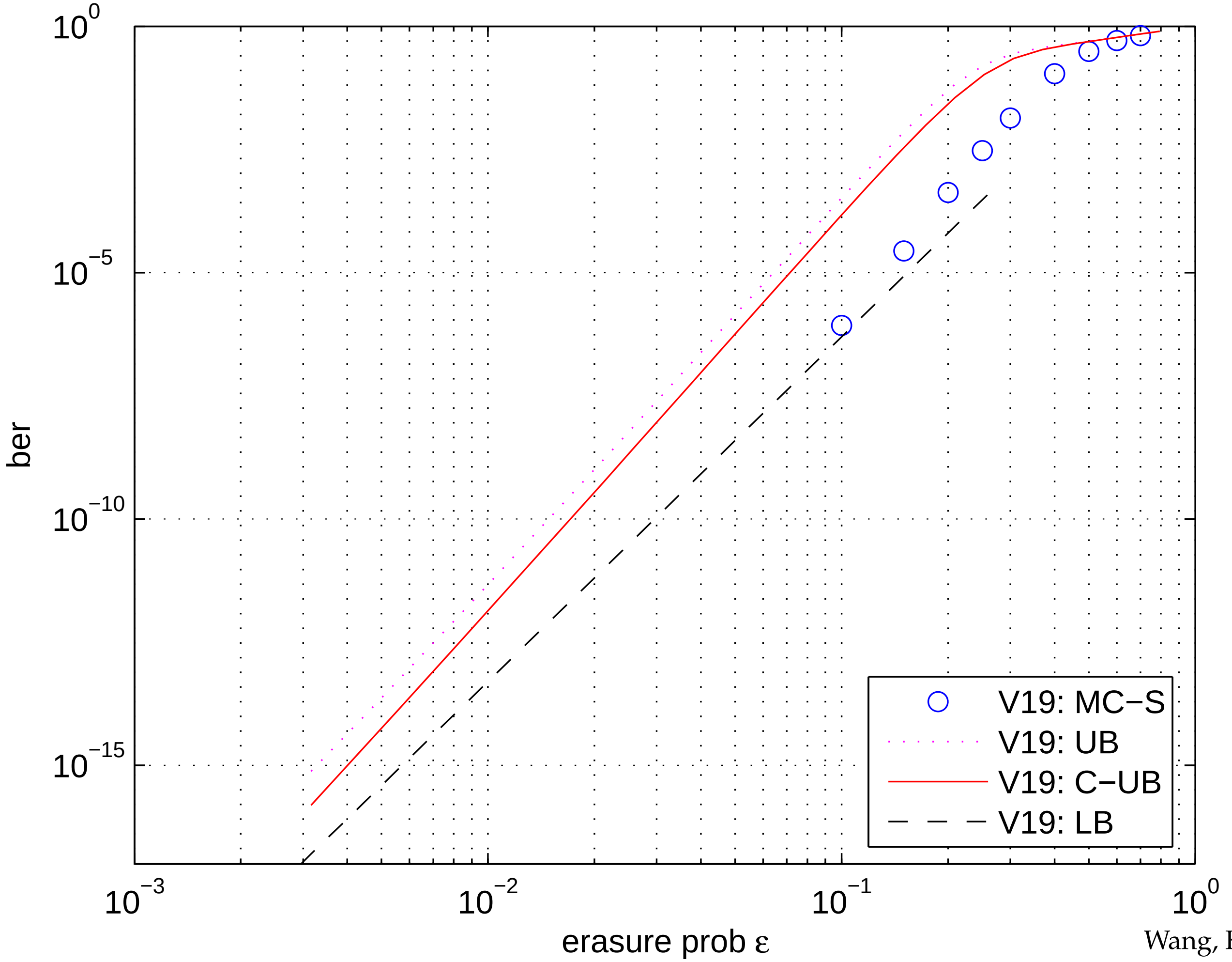
Numerical Experiments (cont'd)

- A fixed, finite LDPC code with var deg. 3, chk deg. 6, $n = 50$:
  - bit 26, (6*,2*)



Numerical Experiments (cont'd)

- A fixed, finite LDPC code with var deg. 3, chk deg. 6, $n = 50$:
  - bit 19, (7*,10 $\longrightarrow$ 5*)



Frame Error Rates and Irregular Codes

- Frame error rate (FER): $f_{FER} = \sum_{i=1}^{n} f_i$.

Frame Error Rates and Irregular Codes

- Frame error rate (FER): $f_{FER} = \sum_{i=1}^{n} f_i$.
- Efficiency:
  - More cycles $\Longrightarrow$ less efficiency. (The Golay code is the worst.)
  - The larger n, the easier for our algorithm.

Frame Error Rates and Irregular Codes

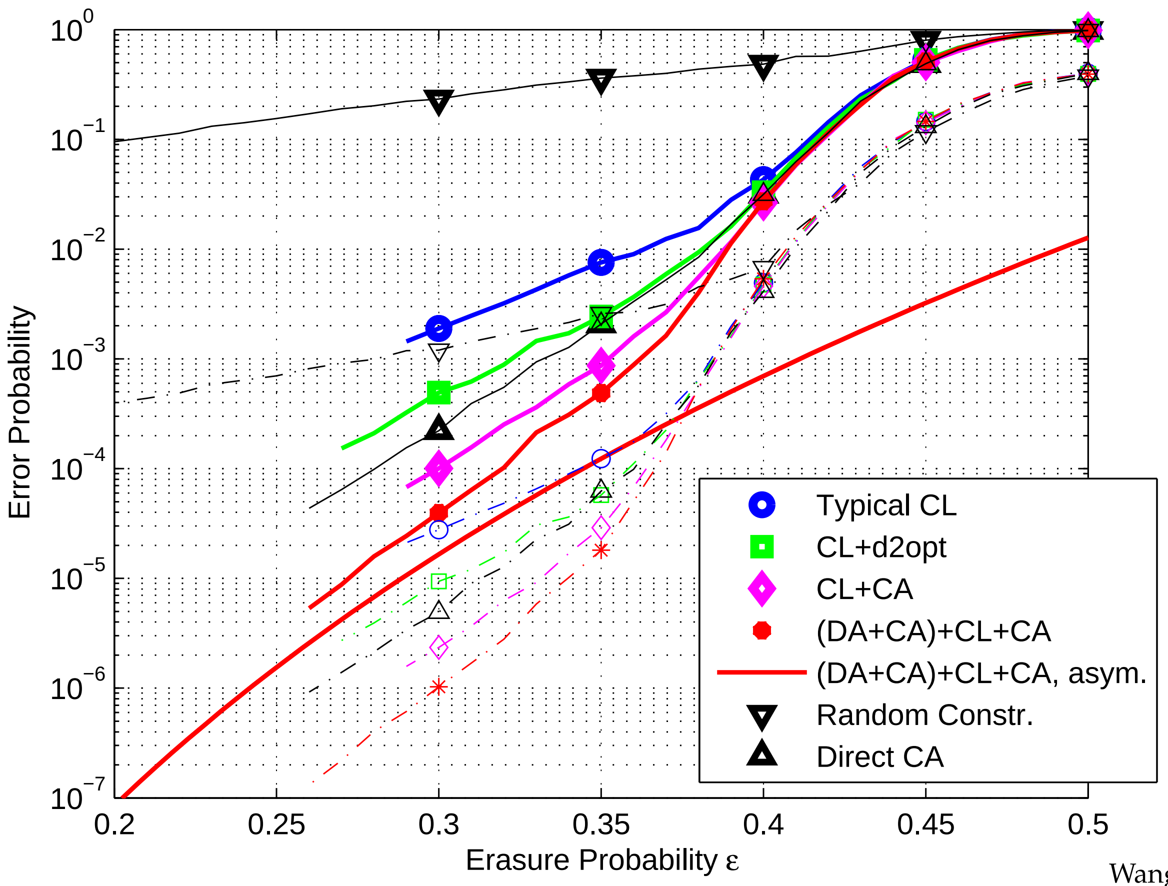
- Frame error rate (FER): $f_{FER} = \sum_{i=1}^{n} f_i$.
- Efficiency:
  - More cycles $\Longrightarrow$ less efficiency. (The Golay code is the worst.)
  - The larger n, the easier for our algorithm.
- Experimental results
  - Consider codes of $n = 500$–1000.
  - For regular codes, $d_{SS} \leq 12$ can be exhausted. ($\binom{512}{12} = 5.9 \times 10^{23}$ trials)
  - The FER of irregular codes suits the algorithm most. $d_{SS} \leq 13$ can be exhausted. ($\binom{512}{13} = 2.5 \times 10^{25}$ trials)

The FER of A Rate 1/2 Irregular Code

$$\lambda(x) = 0.416667x + 0.166667x^2 + 0.416667x^5$$
$$\rho(x) = x^5.$$

61% variable nodes of degree 2

A rate 1/2 $n = 572$ irregular LDPC code, (order, multi)=(13*,104*)



Summary

Upper bounding and exhausting the bad patterns

- An NP-complete problem but feasible for at least short practical LDPC codes.

Summary

Upper bounding and exhausting the bad patterns

- An NP-complete problem but feasible for at least short practical LDPC codes.
- Taking advantages of the tree-like structure.

Summary

Upper bounding and exhausting the bad patterns

- An NP-complete problem but feasible for at least short practical LDPC codes.
- Taking advantages of the tree-like structure.
- Suitable for both FER and ber, punctured codes, unequal sub-channel analysis, non-sparse codes, etc.

Summary

Upper bounding and exhausting the bad patterns

- An NP-complete problem but feasible for at least short practical LDPC codes.
- Taking advantages of the tree-like structure.
- Suitable for both FER and ber, punctured codes, unequal sub-channel analysis, non-sparse codes, etc.
- Our algorithm can be easily modified for trapping sets.

Summary

Upper bounding and exhausting the bad patterns

- An NP-complete problem but feasible for at least short practical LDPC codes.
- Taking advantages of the tree-like structure.
- Suitable for both FER and ber, punctured codes, unequal sub-channel analysis, non-sparse codes, etc.
- Our algorithm can be easily modified for trapping sets.
- A useful tool for finite code optimization.

Summary

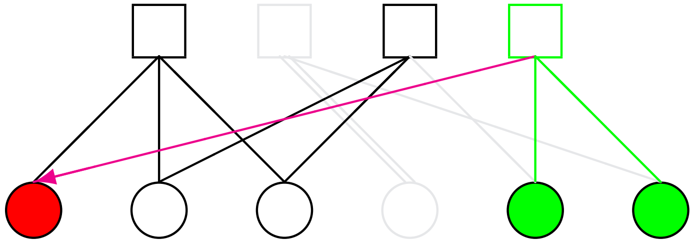
Upper bounding and exhausting the bad patterns

- An NP-complete problem but feasible for at least short practical LDPC codes.
- Taking advantages of the tree-like structure.
- Suitable for both FER and ber, punctured codes, unequal sub-channel analysis, non-sparse codes, etc.
- Our algorithm can be easily modified for trapping sets.
- A useful tool for finite code optimization.
- A starting point for more efficient algorithms.

Thank you for the attention.

Searching for Trapping Sets

- We define the k-out trapping set, namely, the induced graph has k check node of degree 1.



Theorem 5 *Consider a fixed k. Determining the k-out trapping distance is NP-complete.*

- Our algorithm can be modified for trapping set exhaustion.

Some Notes

- Complexity of evaluating UB: $\mathcal{O}(|\mathcal{T}|\log(|\mathcal{T}|))$, even for arbitrarily small ϵ.

Some Notes

- Complexity of evaluating UB: $\mathcal{O}(|\mathcal{T}|\log(|\mathcal{T}|))$, even for arbitrarily small ϵ.
- A pleasant byproduct: an exhaustive list of minimum SS leading to a lower bound tight in both order and multiplicity.

Some Notes

- Complexity of evaluating UB: $\mathcal{O}(|\mathcal{T}|\log(|\mathcal{T}|))$, even for arbitrarily small ϵ.
- A pleasant byproduct: an exhaustive list of minimum SS leading to a lower bound tight in both order and multiplicity.
- Complexity and performance tradeoff.

Some Notes

- Complexity of evaluating UB: $\mathcal{O}(|\mathcal{T}|\log(|\mathcal{T}|))$, even for arbitrarily small ϵ.
- A pleasant byproduct: an exhaustive list of minimum SS leading to a lower bound tight in both order and multiplicity.
- Complexity and performance tradeoff.
- Punctured vs. Shortened Codes

Some Notes

- Complexity of evaluating UB: $\mathcal{O}(|\mathcal{T}|\log(|\mathcal{T}|))$, even for arbitrarily small ϵ.
- A pleasant byproduct: an exhaustive list of minimum SS leading to a lower bound tight in both order and multiplicity.
- Complexity and performance tradeoff.
- Punctured vs. Shortened Codes
- A partitioned approach — a hybrid search

$$\mathsf{E}\{f\} = \sum_j \mathsf{E}\{\mathcal{A}_j\}\mathsf{E}\{f|\mathcal{A}_j\}.$$

Ex: $\mathcal{A}_1 = \{x_3 = 0\}$, $\mathcal{A}_2 = \{x_3 = 1, x_7 = 0\}$, and $\mathcal{A}_3 = \{x_3 = 1, x_7 = 1\}$.

The Tree Converting Algorithm

repeat
    Find the next leaf variable node, say x_j.
    if there exists another non-leaf x_j in $\mathcal{T}$ **then**
        if the youngest common ancestor of the leaf x_j and existing non-leaf x_j, denoted as $\mathsf{yca}(x_j)$, is a check node **then**
            Include the new x_j in $\mathcal{T}$.
        else if $\mathsf{yca}(x_j)$ is a variable node **then**
            Do the pivoting construction.
        end if
    end if
    Construct the immediate children of x_j.
until the size of $\mathcal{T}$ exceeds the preset limit.