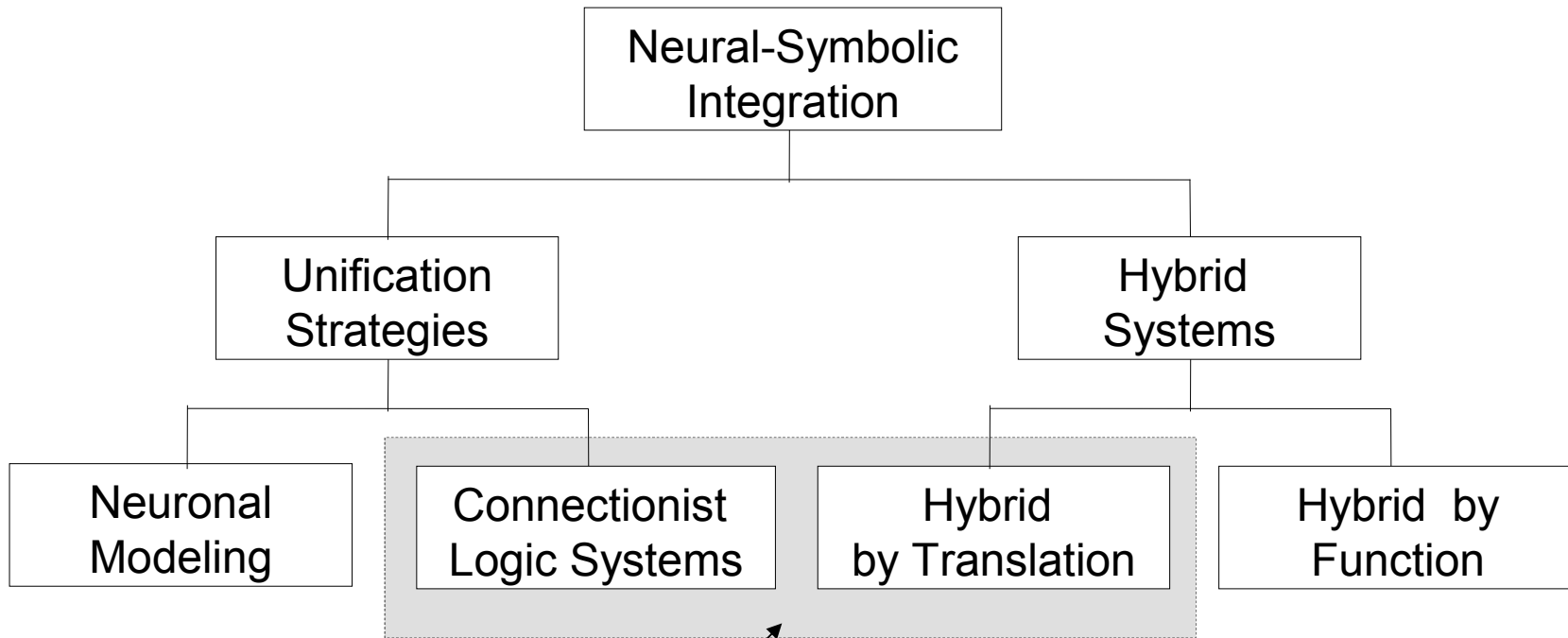


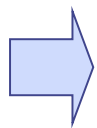
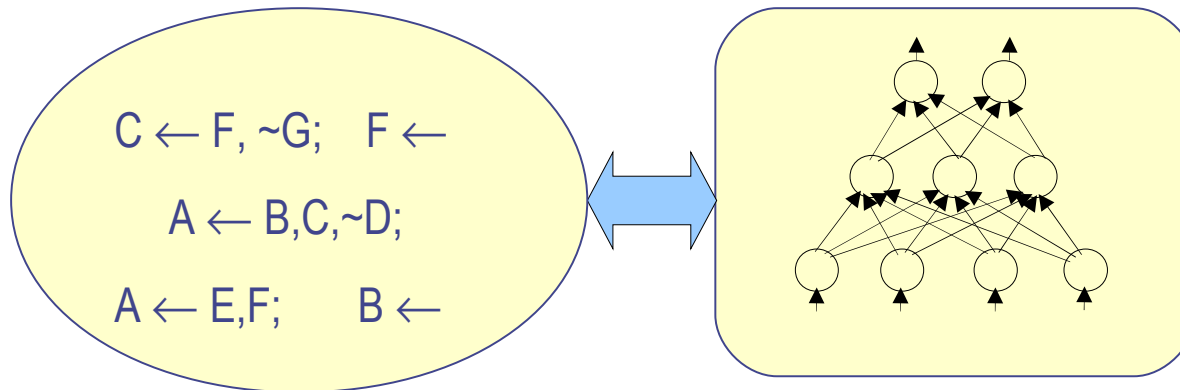
Neural-Symbolic Integration Strategies



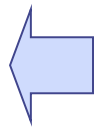
Neural-Symbolic Learning Systems

CILP: Connectionist Inductive Logic Programming System

Objective: To benefit from the integration of Artificial Neural Networks and Symbolic Rules.

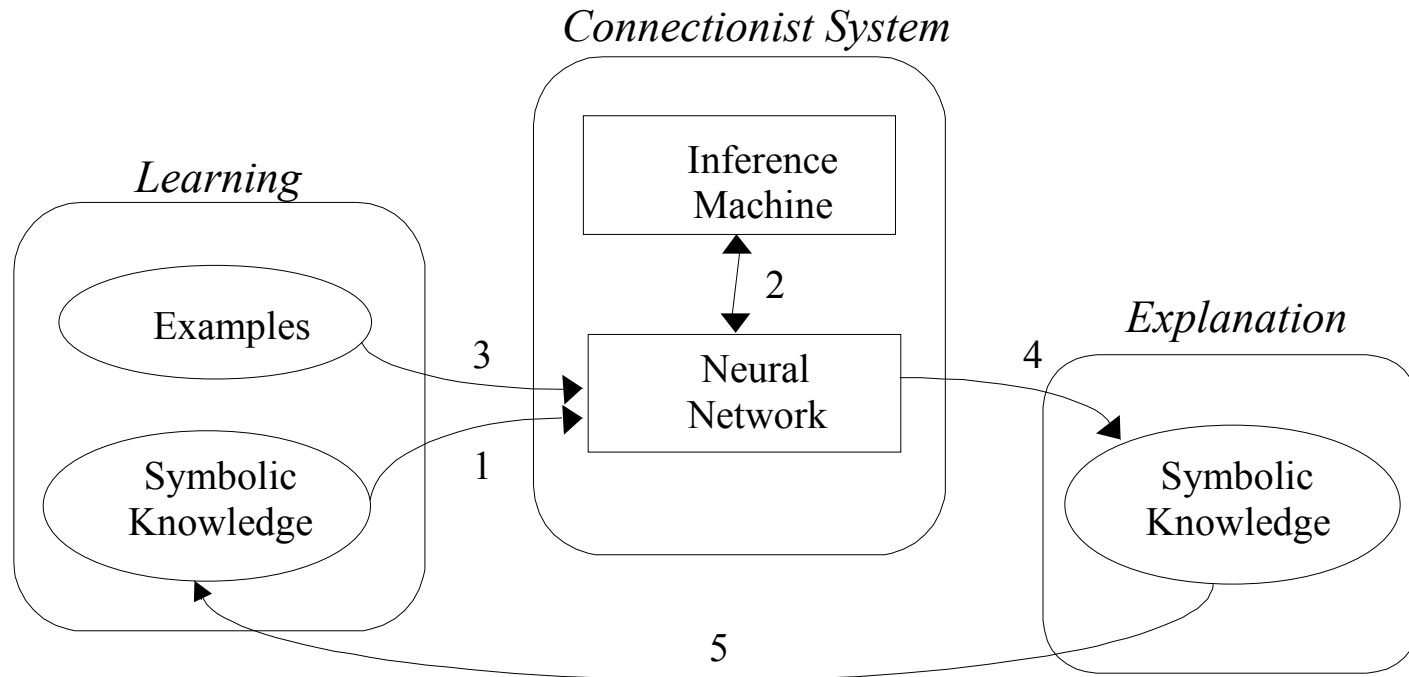


Exploiting Background Knowledge
Explanation Capability



Efficient Learning
Massively Parallel Computation

CILP structure



1. Adding Background Knowledge (BK)
2. Computing BK in Parallel
3. Adding Training with Examples
4. Extracting Knowledge
5. Closing the Cycle

Theory Refinement Contents

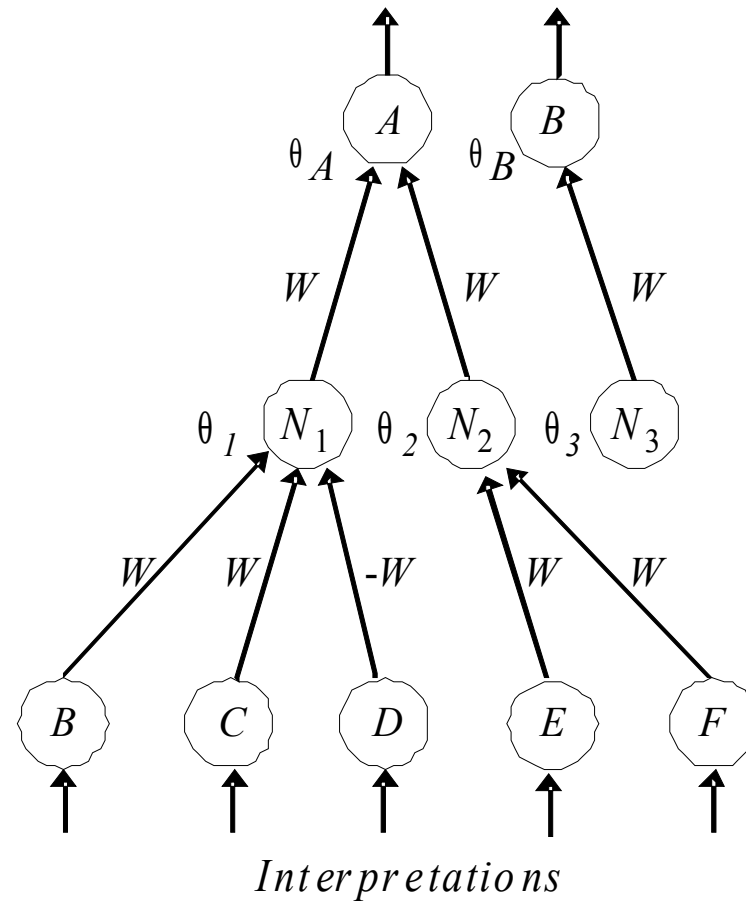
- Inserting Background Knowledge
- Performing Inductive Learning with Examples
- Adding Classical Negation
- Adding Metalevel Priorities
- Experimental Results

Inserting Background Knowledge

Example: $P = \{ A \leftarrow B, C, \sim D;$

$A \leftarrow E, F;$

$B \leftarrow \}$



General clause: $A_0 \leftarrow A_1 \dots A_m, \sim A_{m+1} \dots \sim A_n$

Inserting Background Knowledge

Theorem: For each general logic program P , there exists a feedforward neural network N with exactly one hidden layer and **semi-linear neurons** such that N computes T_P .

Corollary (analogous to [Holldobler and Kalinke 94]): Let P be an acceptable general program. There exists a recurrent neural network N_r with semi-linear neurons such that, starting from an arbitrary initial input, N_r converges to the unique stable model of P .

CILP translation algorithm

Produces neural network N given logic program P
 N can be trained with backpropagation subsequently

Given P with clauses of the form: A if L_1, \dots, L_k

Let p be the number of positive literals in L_1, \dots, L_k

Let m be the number of clauses in P with A in the head

A_{min} denotes the minimum activation for a neuron to be true

A_{max} denotes the maximum activation for a neuron to be false

$$\Theta_h = (1+A_{min}).(k-1).W/2 \quad (\text{threshold of hidden neuron})$$

$$\Theta_A = (1+A_{min}).(1-m).W/2 \quad (\text{threshold of output neuron})$$

$$W > 2 (\ln(1+A_{min}) - \ln(1-A_{min})) / (\max(k,m).(A_{min}-1)+A_{min}+1)$$

$$A_{min} > \max(k,m)-1 / \max(k,m)+1$$

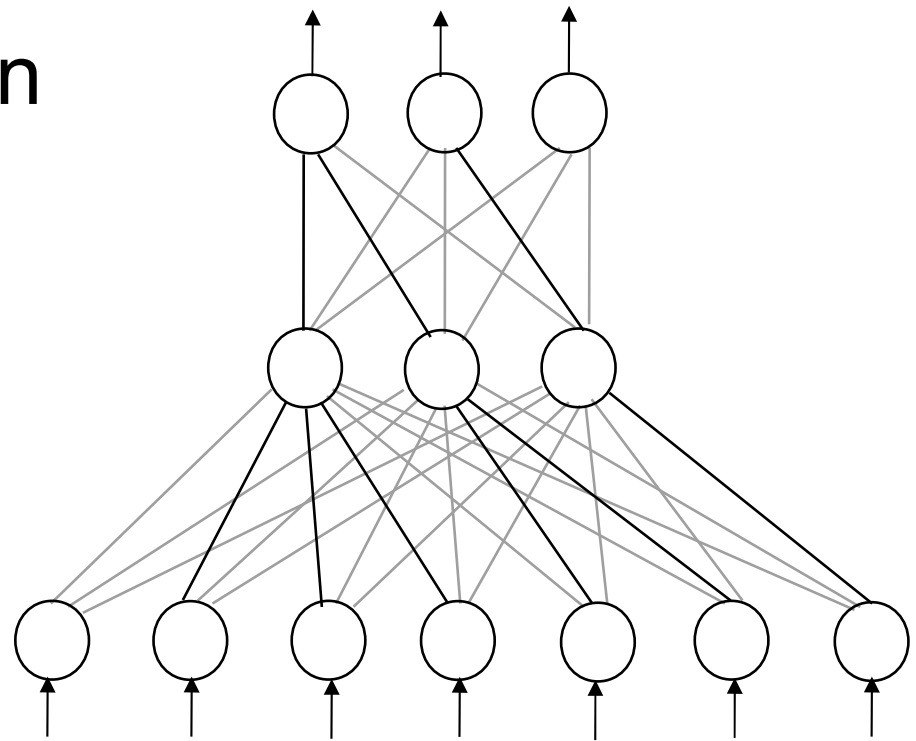
$$A_{max} = -A_{min} \quad (\text{for simplicity})$$

Performing Inductive Learning with Background Knowledge

- Neural Networks may be trained with examples to approximate the operator T_P associated with a Logic Program P .
- A differentiable activation function, e.g. the bipolar semi-linear function $h(x) = (2 / (1 + e^{-x})) - 1$, allows efficient learning with Backpropagation.

Performing Inductive Learning with Background Knowledge

- We add extra input, output and hidden neurons, depending on the application
- We fully-connect the network
- We use Backpropagation



Adding classical negation

General Program:

Cross $\leftarrow \sim$ Train

School bus crosses rail line in the absence of proof of approaching train

Extended Program:

Cross $\leftarrow \neg$ Train

School bus crosses rail line if there is proof of no approaching train

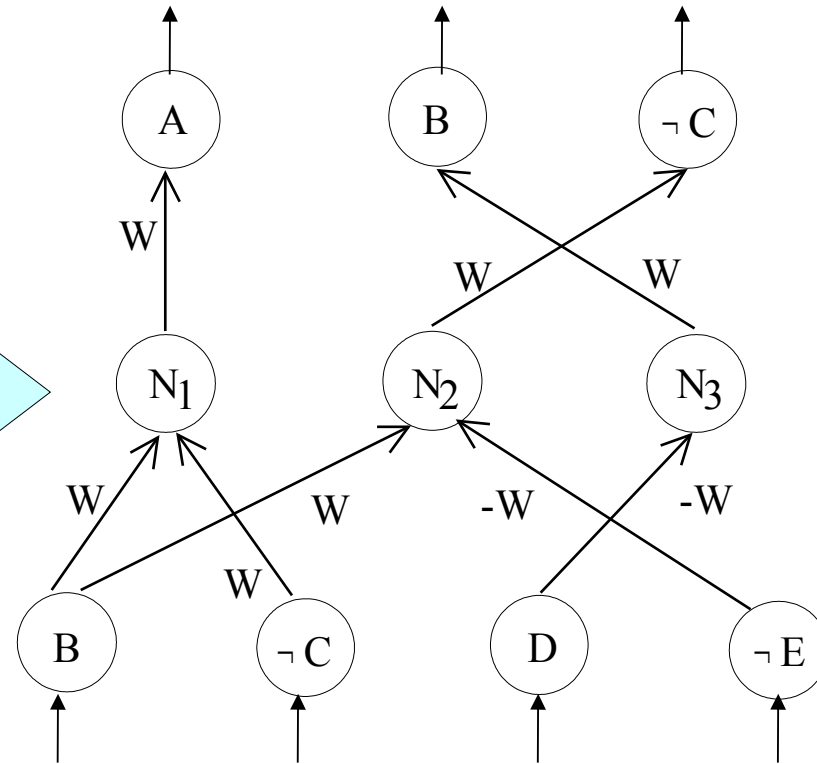
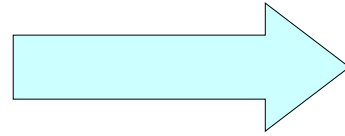
Extended clause: $L_0 \leftarrow L_1 \dots L_m, \sim L_{m+1} \dots \sim L_n$

The Extended CILP System

$r_1: A \leftarrow B, \neg C;$

$r_2: \neg C \leftarrow B, \sim \neg E;$

$r_3: B \leftarrow \sim D$



Adding Classical Negation

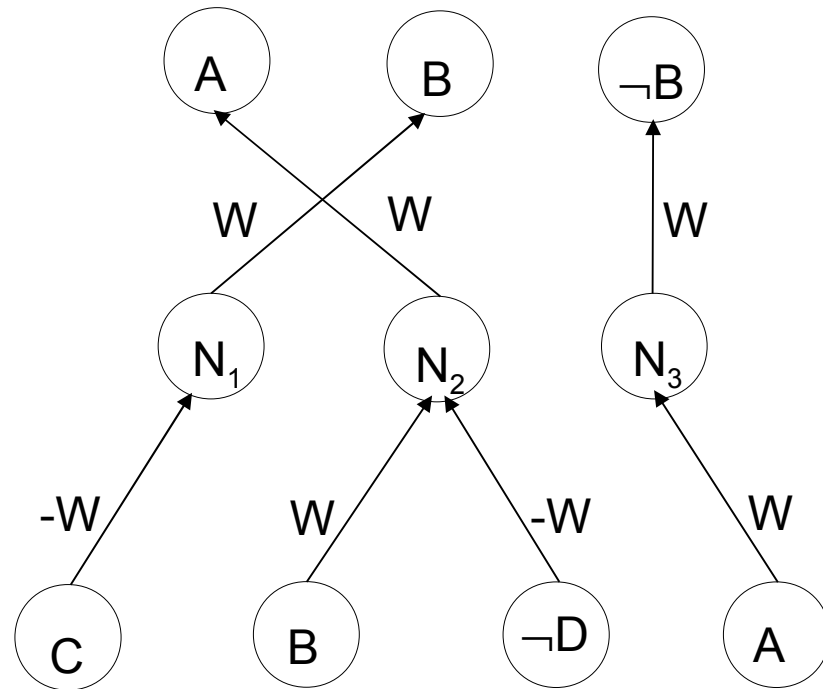
Theorem: For each extended logic program P , there exists a feedforward neural network N with exactly one hidden layer and semi-linear neurons such that N computes T_P .

Corollary: Let P be a consistent acceptable extended program. There exists a recurrent neural network N_r with semi-linear neurons such that, starting from an arbitrary initial input, N_r converges to the unique answer set of P .

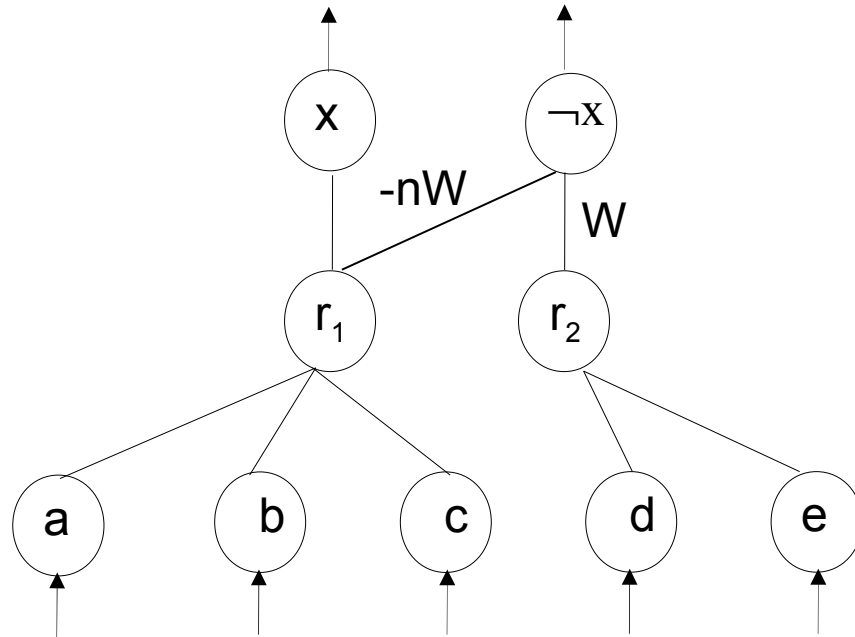
Computing with Classical Negation

Example: $P = \{ B \leftarrow \sim C;$
 $A \leftarrow B, \sim \neg D;$
 $\neg B \leftarrow A \}$.

Recurrently connected, N
converges to stable state:
 $\{A = \text{true}, B = \text{true}, \neg B = \text{true},$
 $C = \text{false}, \neg D = \text{false}\}$



Adding Metalevel Priorities



$r_1 > r_2 =$ “ x is preferred over $\neg x$ ”, i.e.

when r_1 fires it should block the output of r_2

Learning Metalevel Priorities

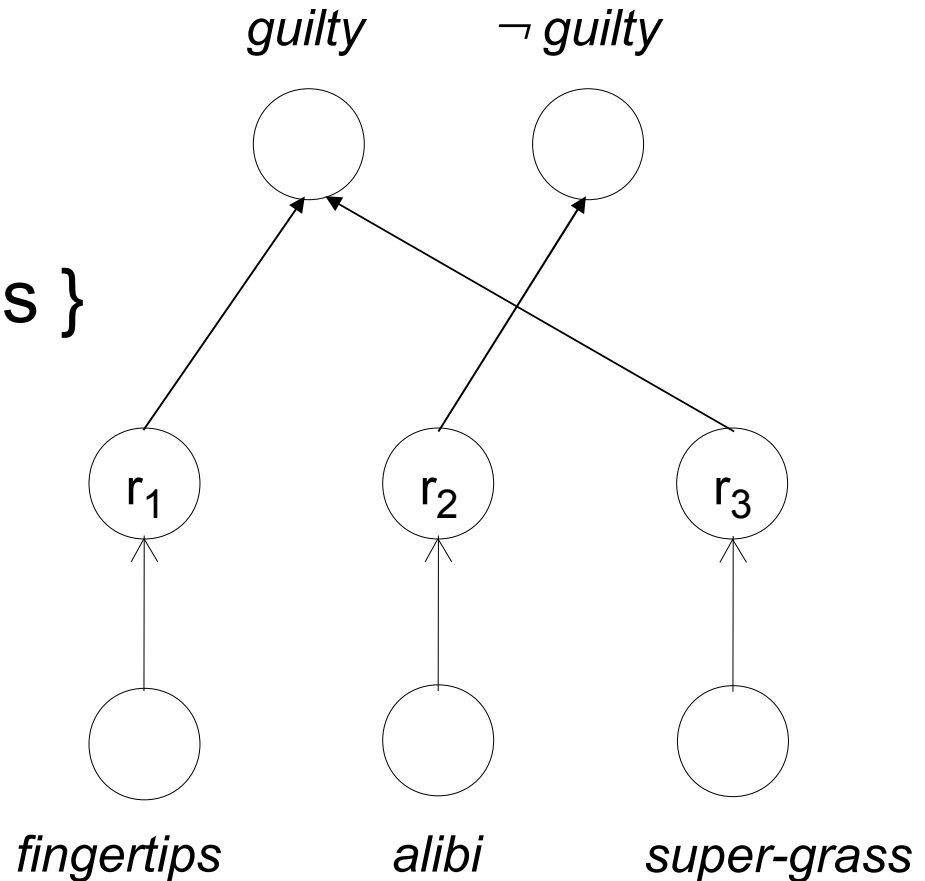
$P = \{ r_1: \text{guilty} \leftarrow \text{fingertips},$
 $r_2: \neg\text{guilty} \leftarrow \text{alibi},$
 $r_3: \text{guilty} \leftarrow \text{supergrass} \}$

$r_1 > r_2 > r_3$

Training examples include:

$[(-1, 1, *), (-1, 1)]$

$[(1, *, *), (1, -1)]$



where * means “*don't care*”

Learning Metalevel Priorities

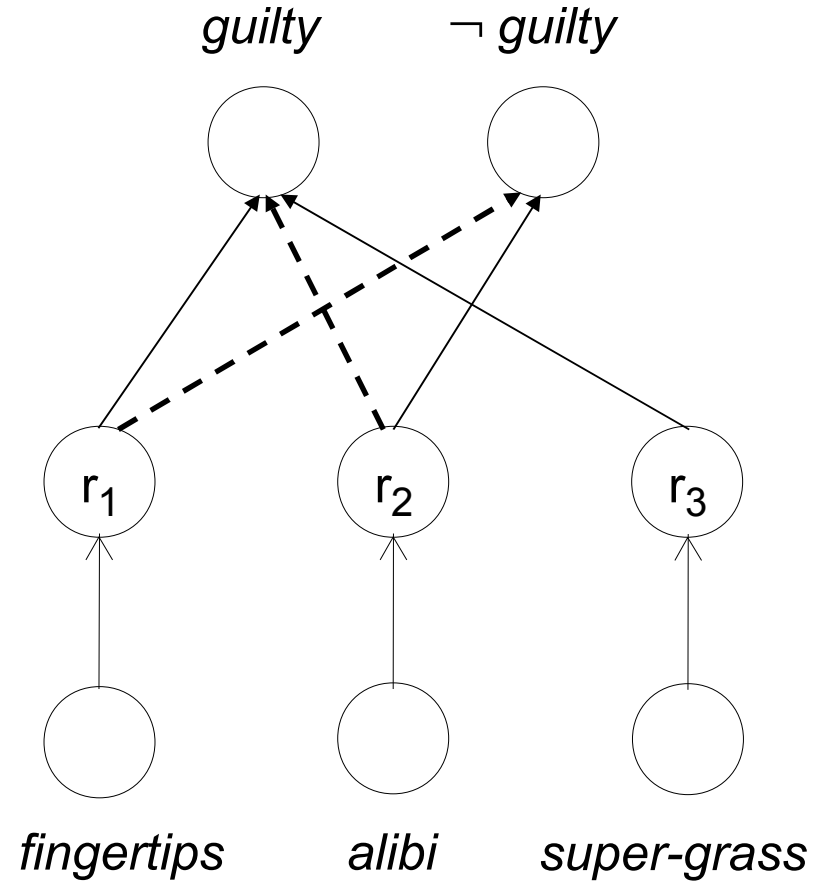
$$W_{\text{guilty}, r1} = 3.97,$$

$$W_{\text{guilty}, r2} = -1.93, \quad W_{\neg\text{guilty}, r1} = -1.93$$

$$W_{\text{guilty}, r3} = 1.94, \quad W_{\neg\text{guilty}, r2} = 1.94$$

$$W_{\neg\text{guilty}, r3} = 0.00$$

$$\theta_{\text{guilty}} = -1.93, \quad \theta_{\neg\text{guilty}} = 1.93$$



$r1 > r2 > r3 ?$

Setting Linearly Ordered Theories

$$W_{\text{guilty},r3} = W$$

$$W_{\text{guilty},r2} = -W + \delta$$

$$\begin{aligned} W_{\text{guilty},r1} &= W_{\text{guilty},r3} - W_{\text{guilty},r2} + \delta \\ &= 2W \end{aligned}$$

If $W = 2, \delta = 0.01$:

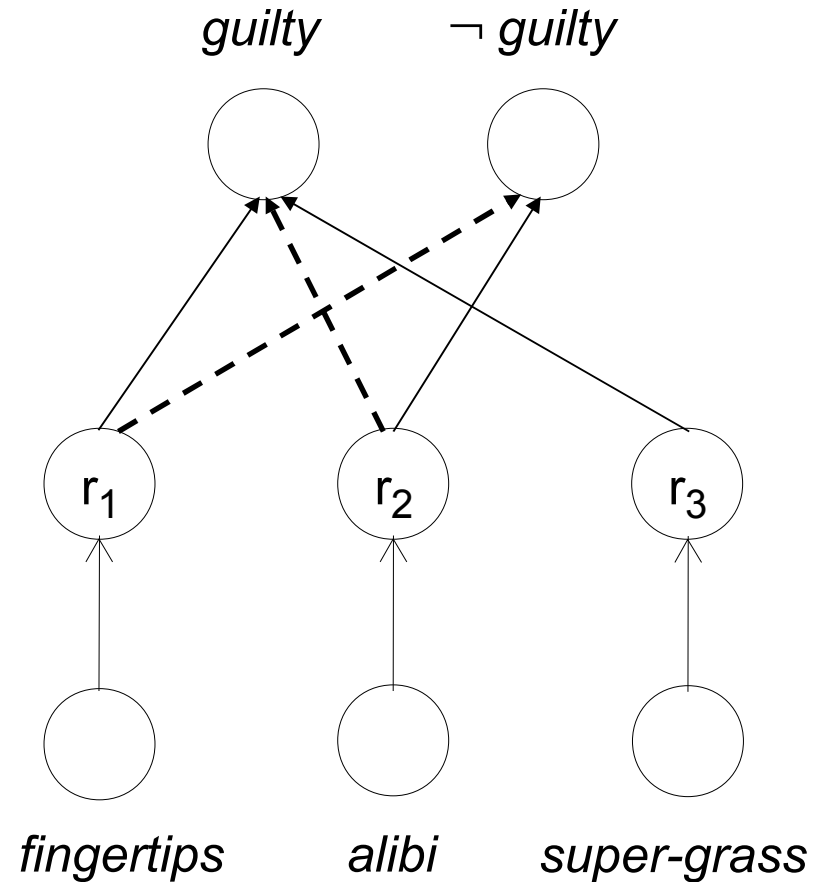
$$W_{\text{guilty},r3} = 2$$

$$W_{\text{guilty},r2} = -1.99$$

$$W_{\text{guilty},r1} = 4$$

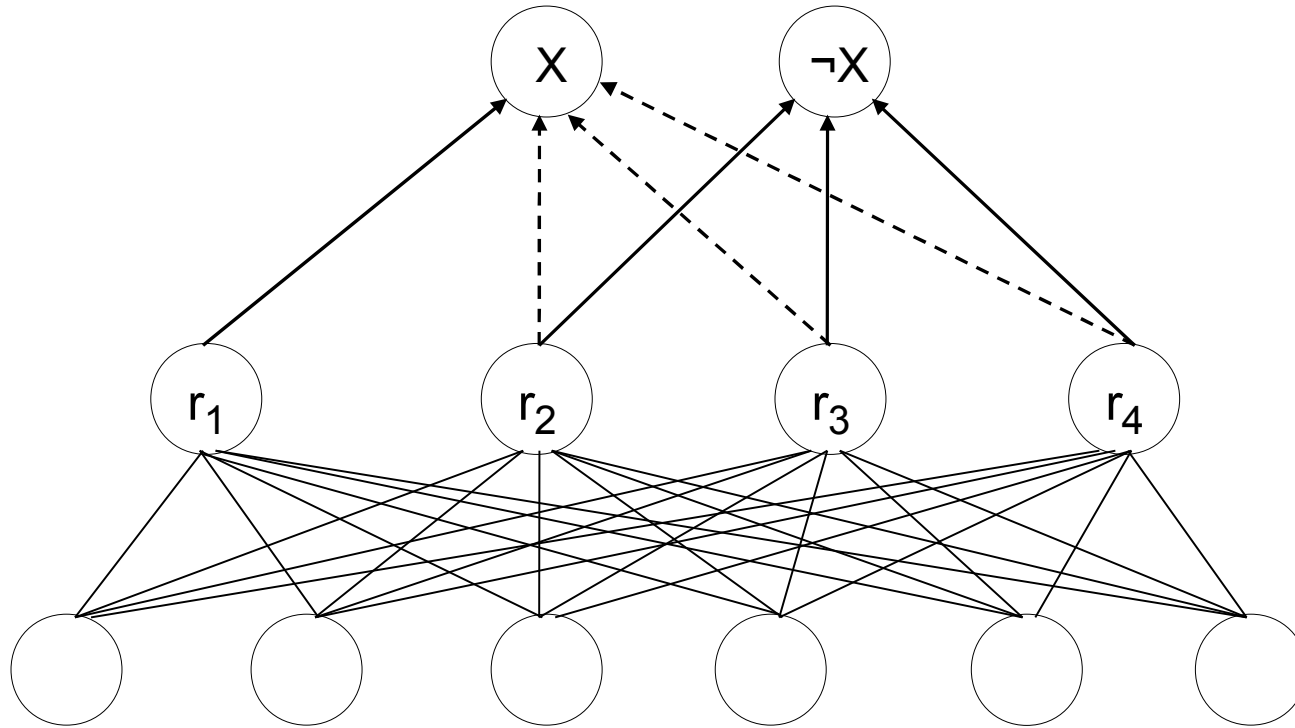
$$-3 < \theta_{\text{guilty}} < 1$$

$$-1 < \theta_{\neg\text{guilty}} < 3$$



$$r1 > r2 > r3$$

Partially Ordered Theories



$r_2 > r_1$
 $r_3 > r_1$
 $r_4 > r_1$

Each of r_2 , r_3 and r_4 should block the conclusion of r_1

Problematic Case

layEggs(platypus) monotreme(platypus)

hasFur(platypus) hasBill(platypus)

r1: mammal(x) \leftarrow monotreme(x)

r2: mammal(x) \leftarrow hasFur(x)

r1 > r3

r3: \neg mammal(x) \leftarrow layEggs(x)

r2 > r4

r4: \neg mammal(x) \leftarrow hasBill(x)

Cannot have r1 > r3, r2 > r4 without also
having r1 > r4 and r2 > r3

CILP Experimental Results

- Test Set Performance (how well it generalises)
- Test Set Performance over small/increasing training sets (how important BK is)
- Training Set Performance (how fast it trains)

CILP Experimental Results

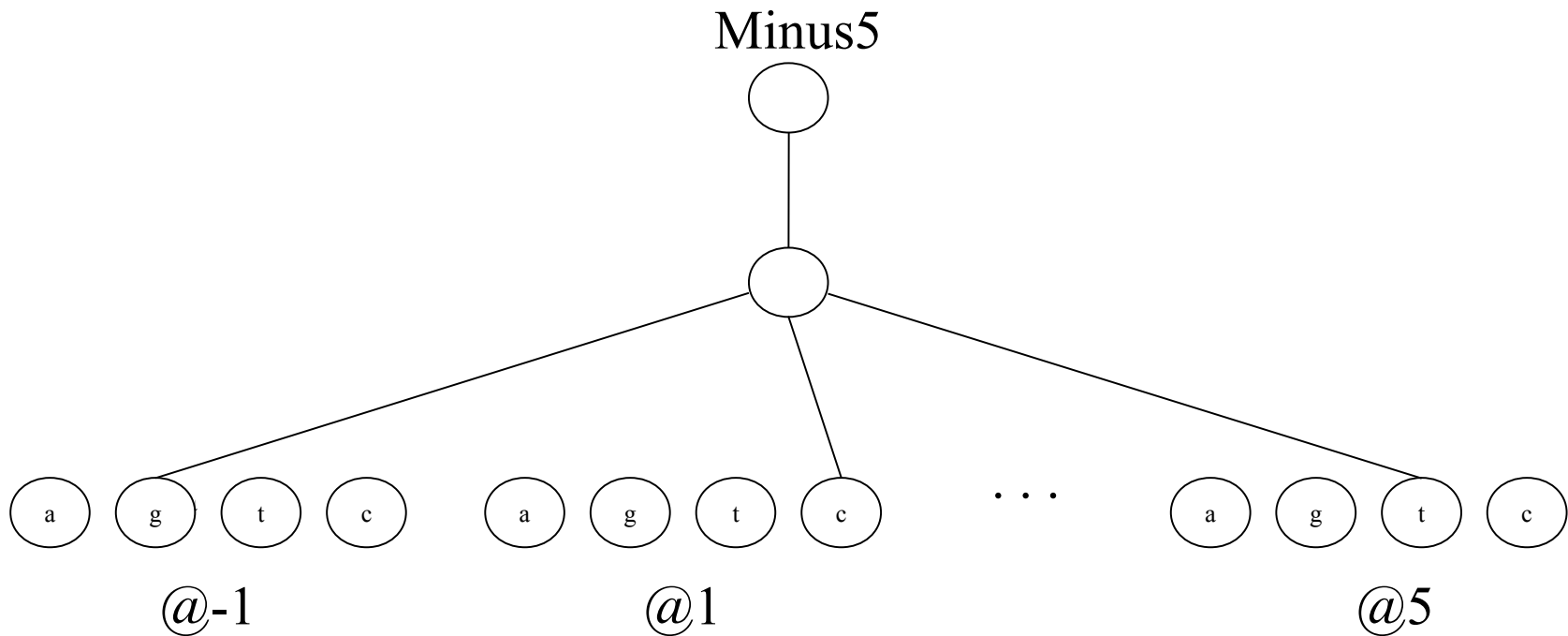
- Promoter Recognition:
 - ♦ A short DNA sequence that precedes the beginning of genes.

Background Knowledge

Promoter ← Contact, Conformation	
Contact ← Minus10, Minus35	
Minus 10 ← @ -14 'tataat'	Minus 10 ← @ -13 'ta', @ -10 'a', @ -8 't'
Minus 10 ← @ -13 'tataat'	Minus 10 ← @ -12 'ta', @ -7 't'
Minus 35 ← @ -37 'cttgac'	Minus 35 ← @ -36 'ttgac'
Minus 35 ← @ -36 'ttgaca'	Minus 35 ← @ -36 'ttg', @ -32 'ca'
Conformation ← @ -45 'aa', @ -41 'a'	
Conformation ← @ -45 'a', @ -41 'a', @ -28 'tt', @ -23 't', @ -21 'aa', @ -17 't', @ -15 't', @ -4 't'	
Conformation ← @ -49 'a', @ -44 't', @ -27 't', @ -22 'a', @ -18 't', @ -16 'tg', @ -1 'a'	
Conformation ← @ -47 'caa', @ -43 'tt', @ -40 'ac', @ -22 'g', @ -18 't', @ -16 'c', @ -8 'gcgcc', @ -2 'cc'	

An Example Bioinformatics Rule

Minus5 \leftarrow @-1'gc', @5't'

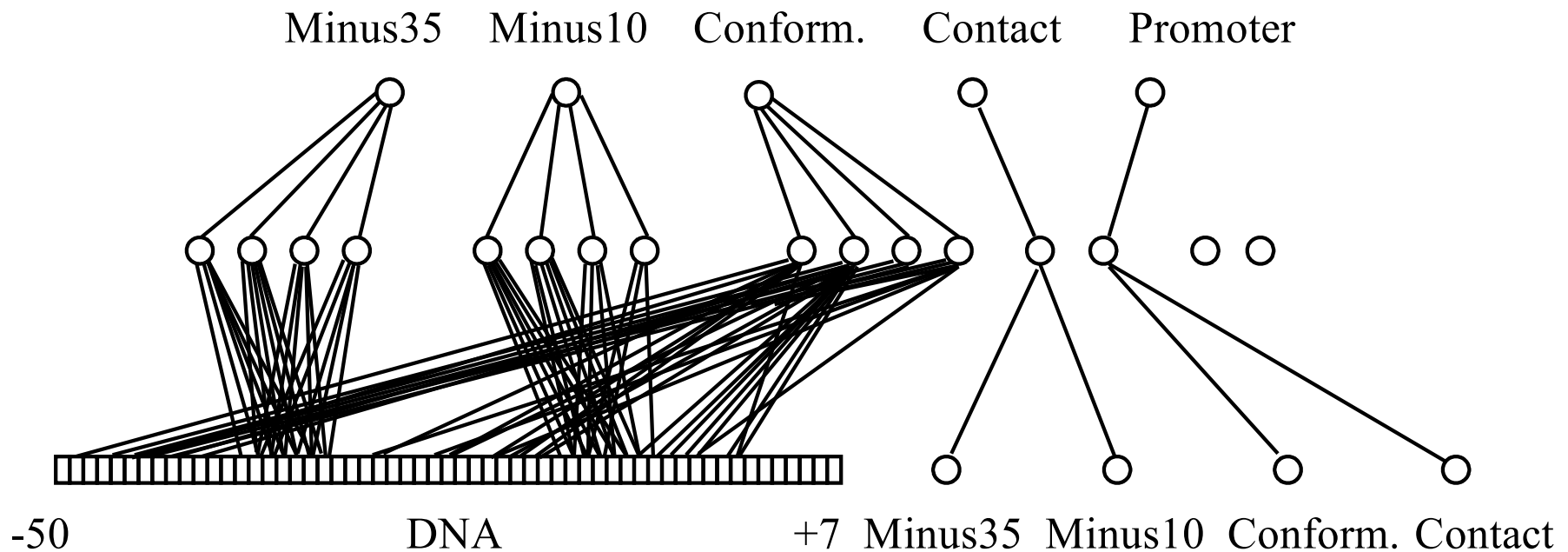


Promoter Recognition

53 examples of promoters

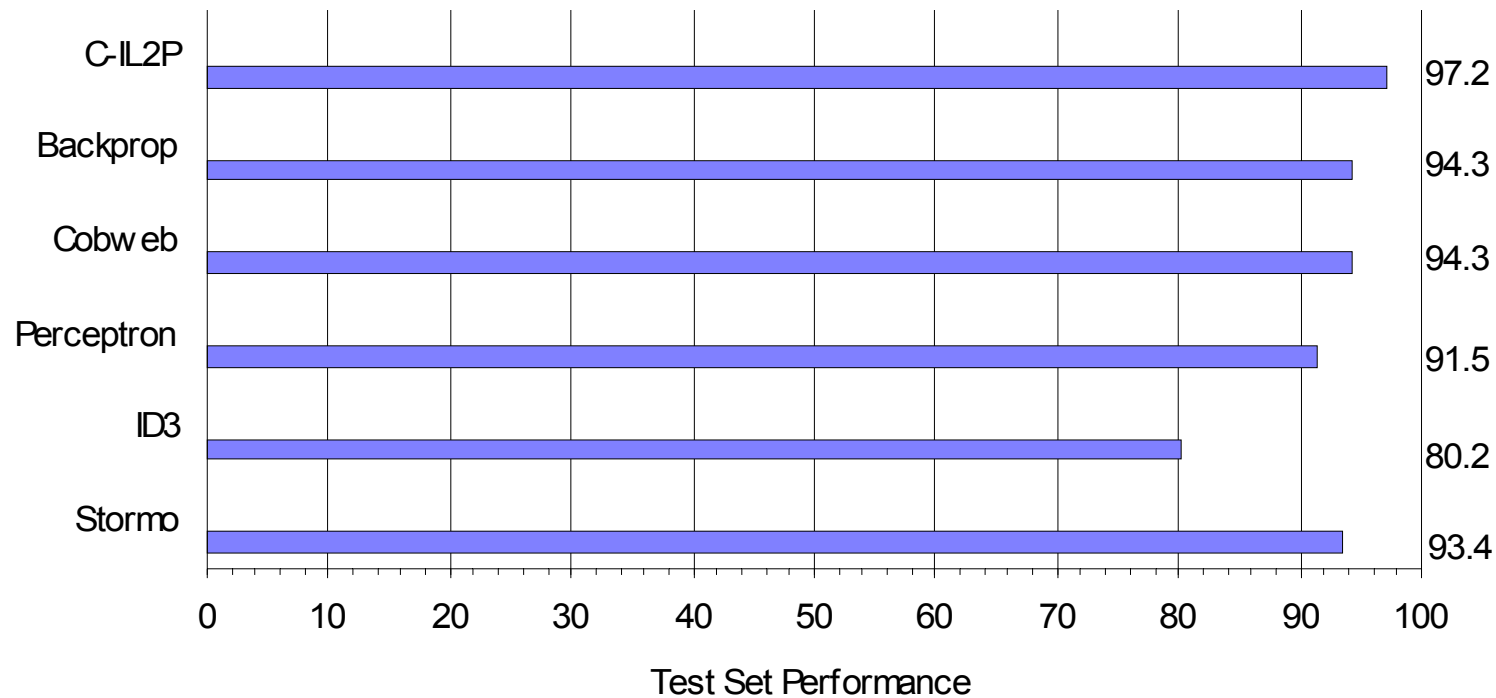
53 examples of non-promoters

Initial Topology of the Network



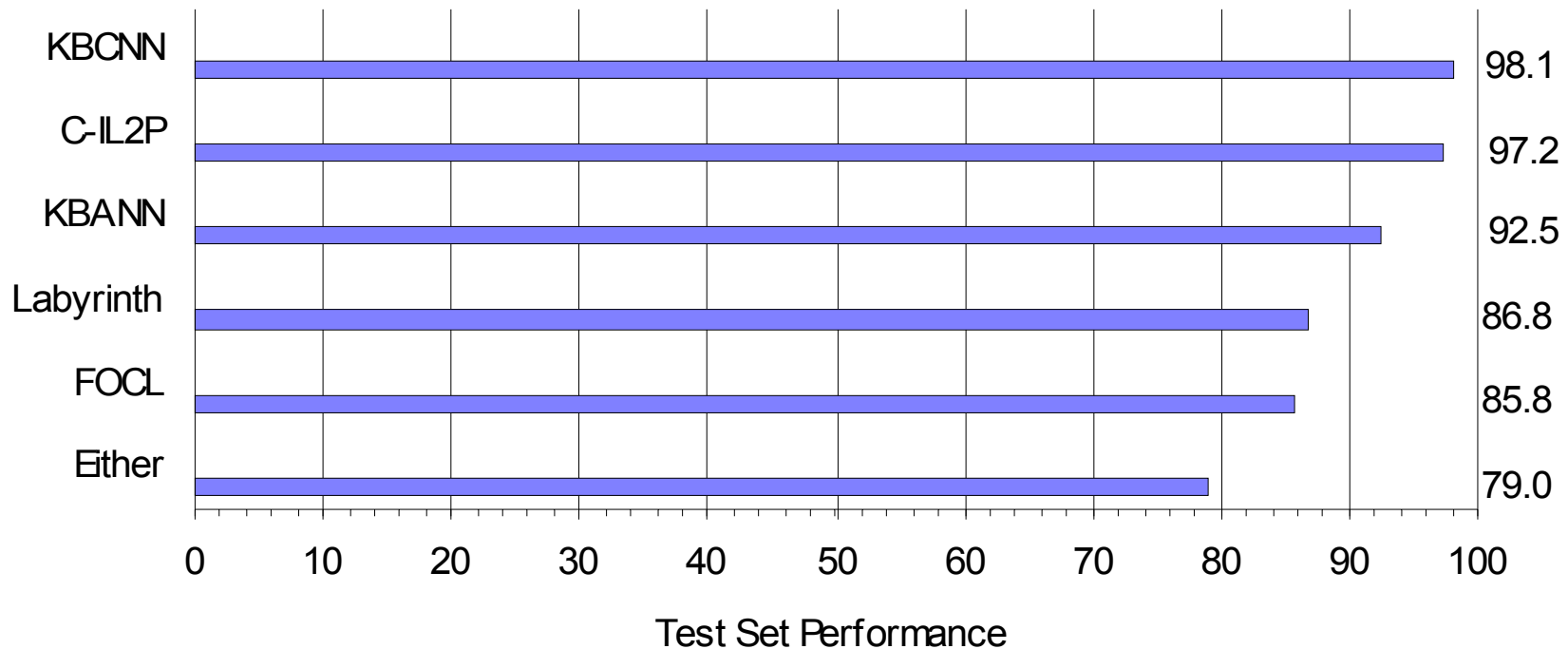
Test Set Performance (promoter recognition)

- Comparison with systems that learn from examples only (i.e. no BK)



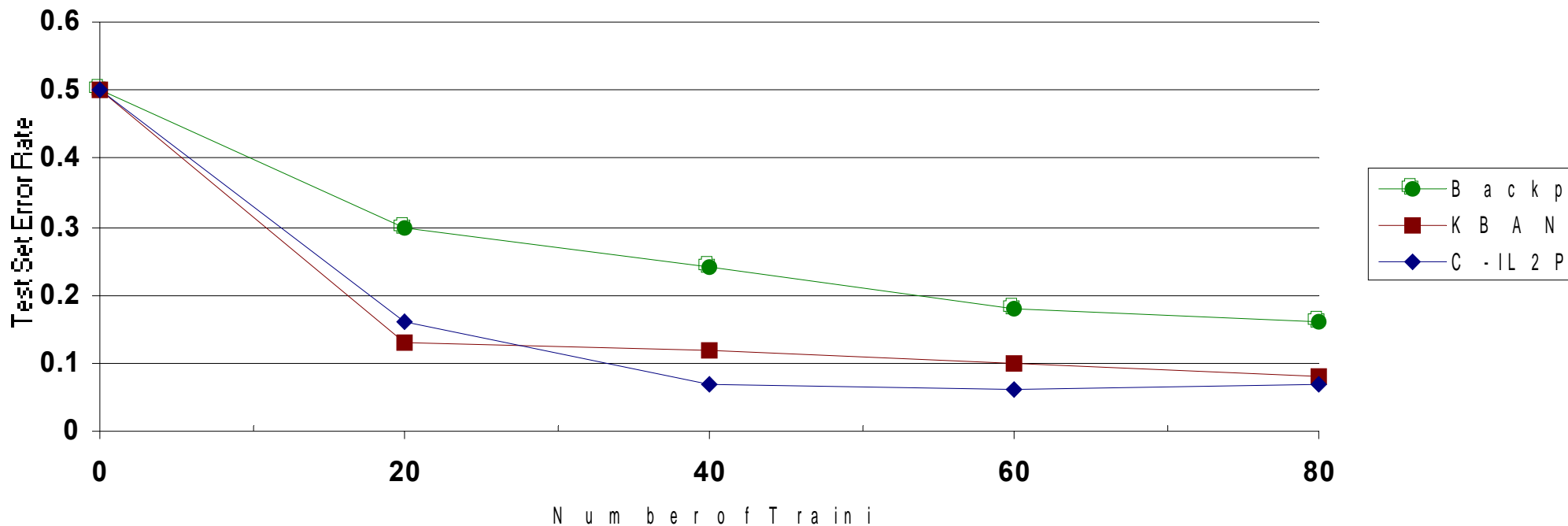
Test Set Performance (promoter recognition)

- Comparison with systems that learn from examples and background knowledge



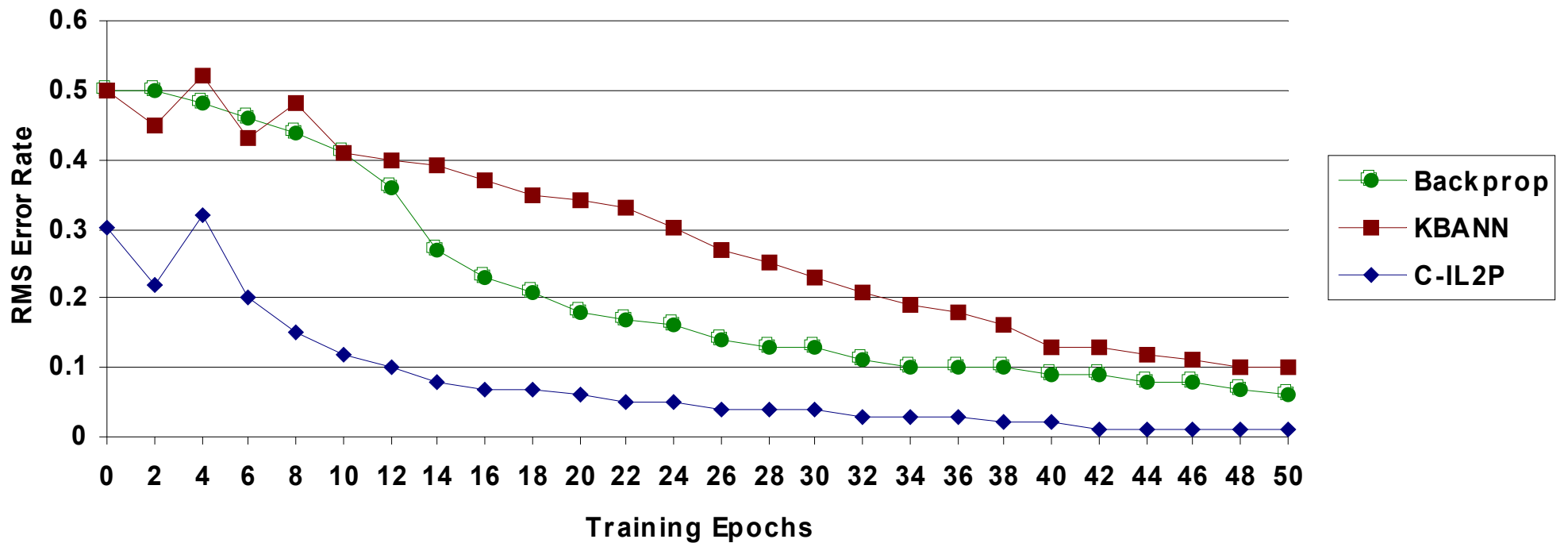
Test set performance on small/increasing training sets

- Promoter recognition: comparison with Backprop and KBANN



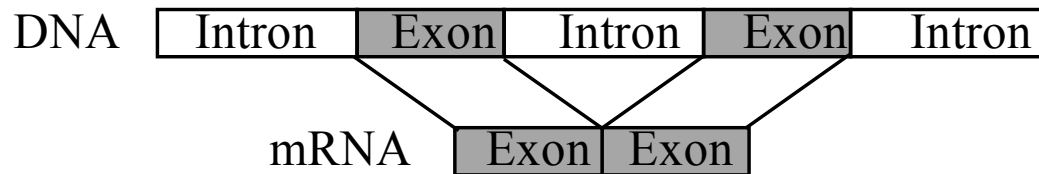
Training Set Performance (promoter recognition)

- Comparison with Backprop and KBANN



CILP Experimental Results

- Splice Junction Determination
 - ♦ Points on a DNA sequence at which the cell removes superfluous DNA during the process of protein creation.



Background Knowledge

EI ← @ -3 'aaggtagt', ~EIStop	EI ← @ -3 'caggtagt', ~EIStop	
EI ← @ -3 'aaggtgagt', ~EIStop	EI ← @ -3 'caggtgagt', ~EIStop	
EI-Stop ← @ -3 'taa'	EI-Stop ← @ -4 'taa'	EI-Stop ← @ -5 'taa'
EI-Stop ← @ -3 'tag'	EI-Stop ← @ -4 'tag'	EI-Stop ← @ -5 'tag'
EI-Stop ← @ -3 'tga'	EI-Stop ← @ -4 'tga'	EI-Stop ← @ -5 'tga'
IE ← @ -3 'tagg', Piramidal, ~IEStop	IE ← @ -3 'cagg', Piramidal, ~IEStop	
Piramidal ← @ -15 'tttttttt'	Piramidal ← @ -15 'cccccccc'	
IE-Stop ← @ 1 'taa'	IE-Stop ← @ 2 'taa'	IE-Stop ← @ 3 'taa'
IE-Stop ← @ 1 'tag'	IE-Stop ← @ 2 'tag'	IE-Stop ← @ 3 'tag'
IE-Stop ← @ 1 'tga'	IE-Stop ← @ 2 'tga'	IE-Stop ← @ 3 'tga'

Splice-Junction Determination

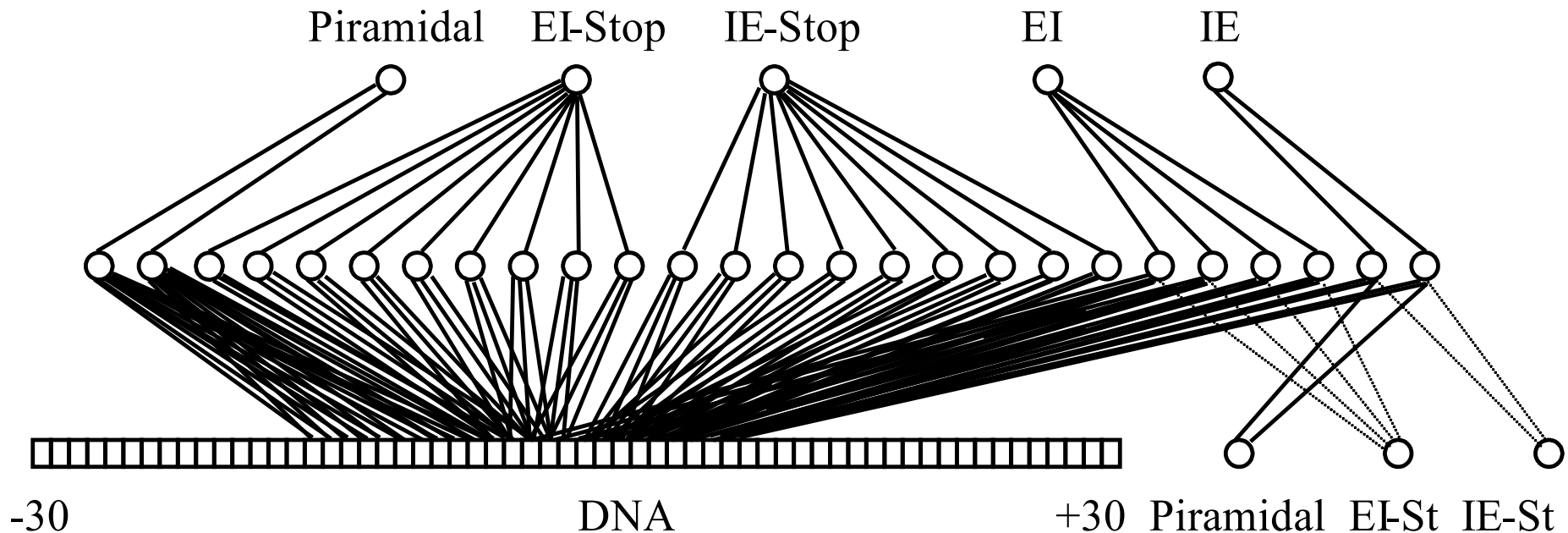
3190 examples:

25 % examples of E/I boundaries

25 % examples of I/E boundaries

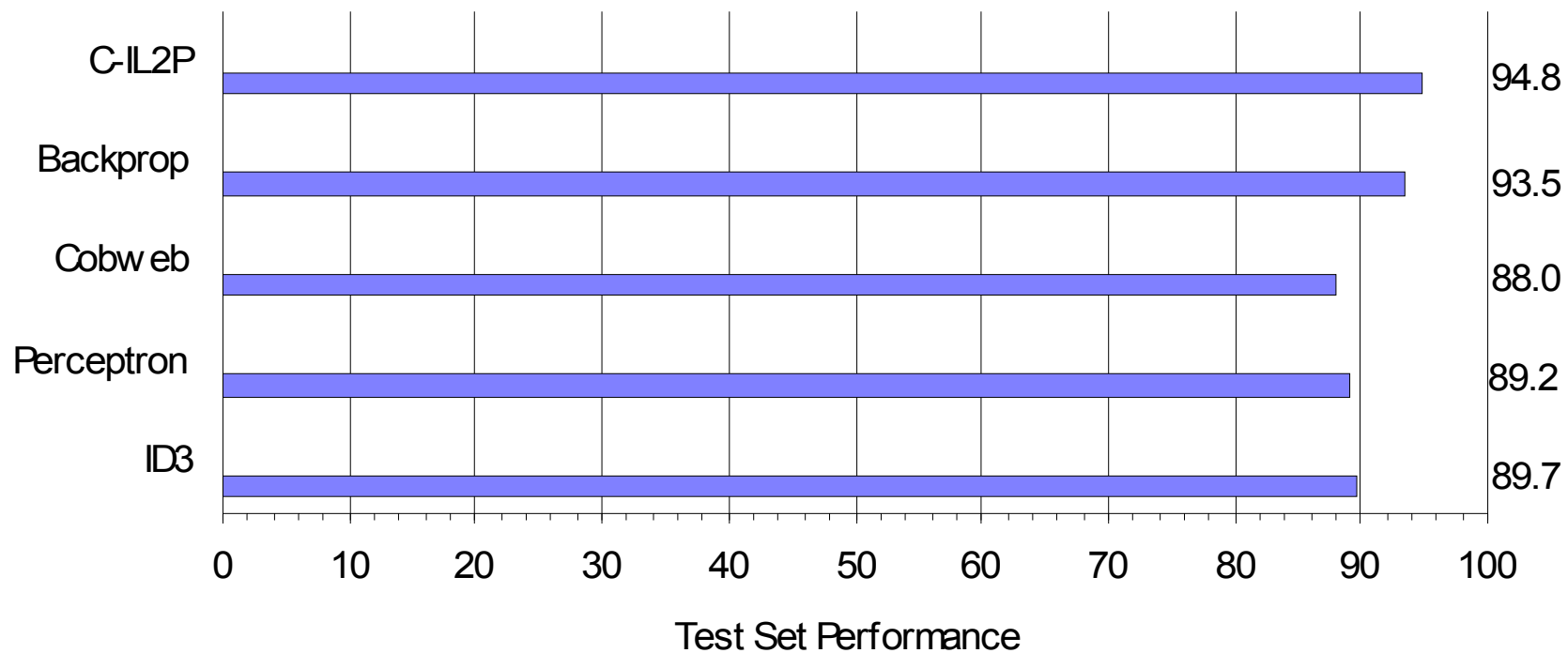
50 % of non-examples

Initial Topology of the Network



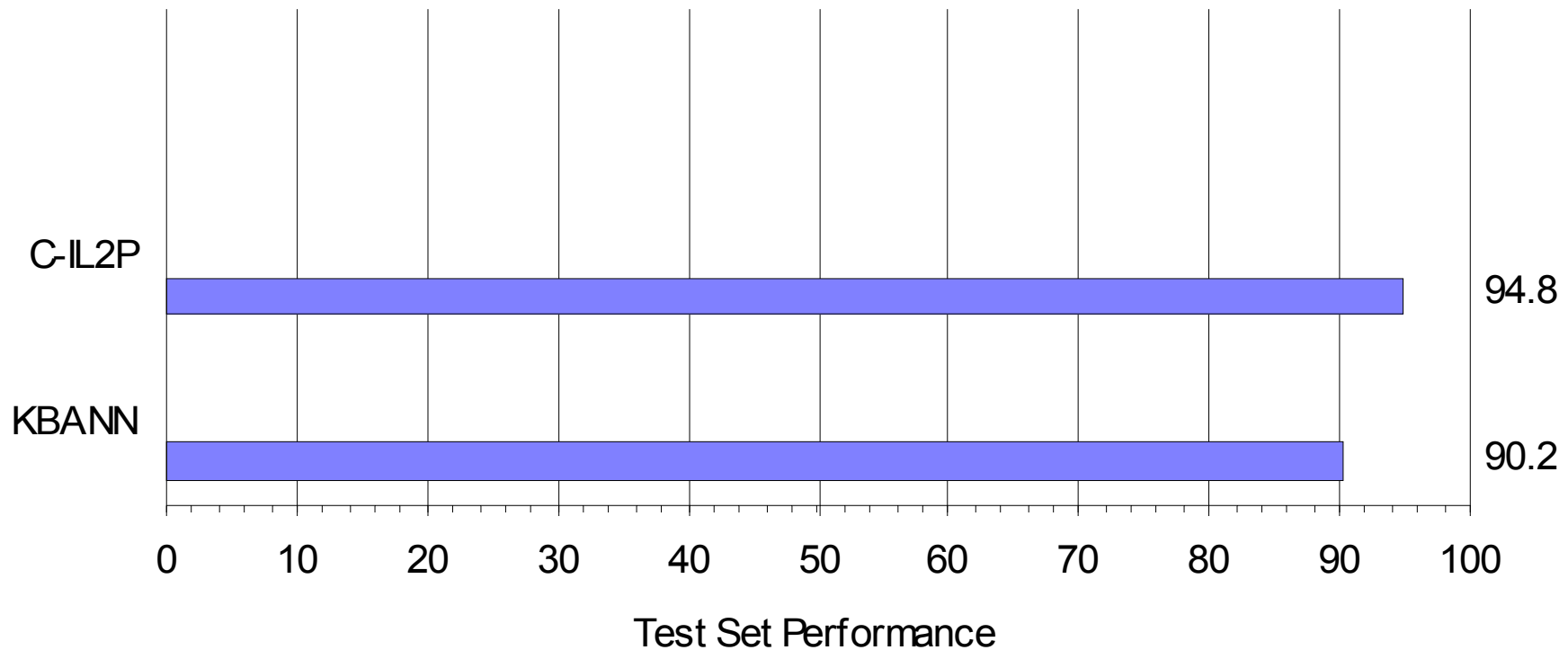
Test Set Performance (Splice-Junction Determination)

- Comparison with systems that learn from examples only (i.e. no BK)



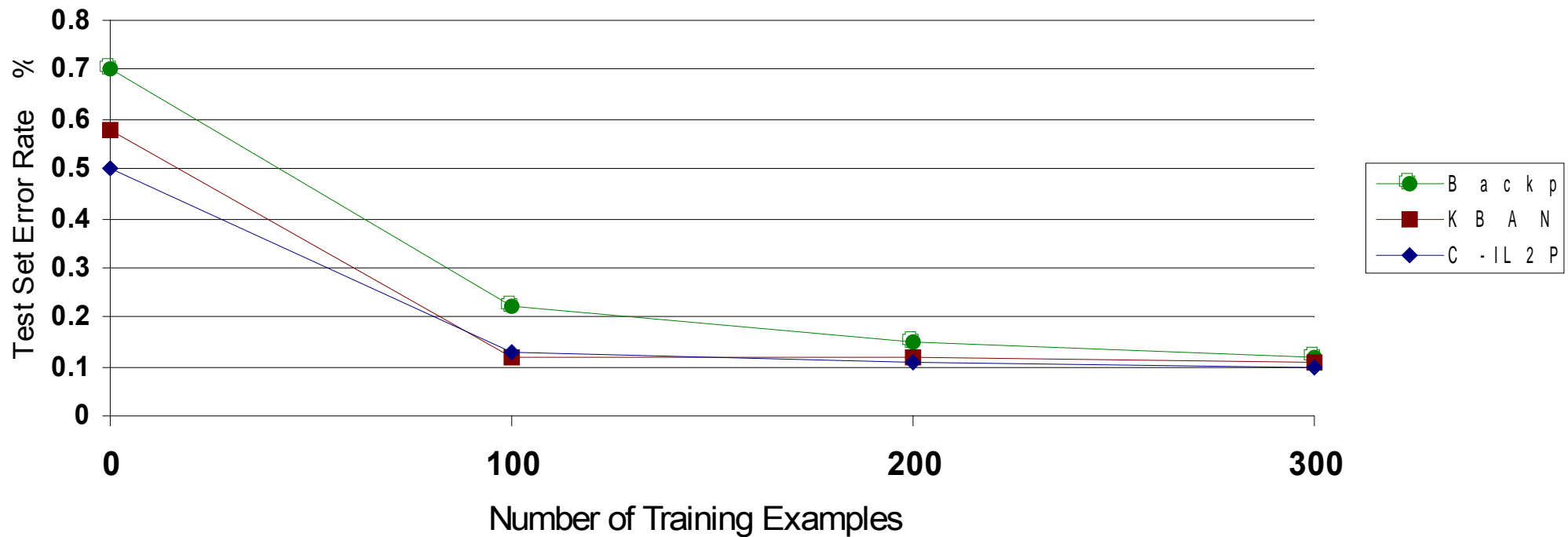
Test Set Performance (Splice-Junction Determination)

- Comparison with systems that learn from examples and background knowledge



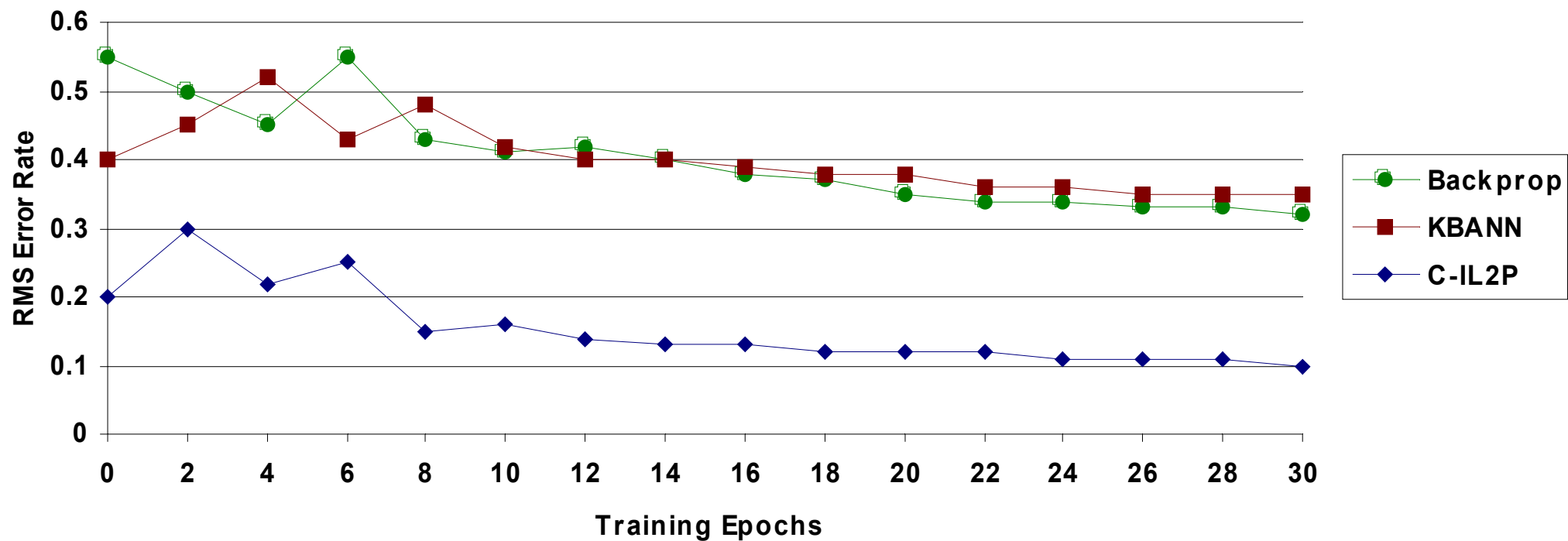
Test set performance on small/increasing training sets

- Splice-junction determination: comparison with Backprop and KBANN



Training Set Performance (Splice-junction determination)

- Comparison with Backprop and KBANN



Experimental Results

summary

- CILP's test set performance is comparable with Backpropagation and KBANN
- CILP's test set performance in the presence of few training examples is better than Backprop and comparable with KBANN
- CILP's training set performance is superior than Backprop and KBANN

Sources of CILP strength

- CILP uses Backpropagation
- CILP uses Background Knowledge
- CILP's translation of BK into N is compact and correct:
 - Single-hidden layer network
 - Provably sound translation algorithm

Theory Refinement Summary

- The combination of theory and data learning provides more effective machine learning systems.
- Single hidden layer neural networks can be used to represent and learn extended logic programs.
- Preference relations can be encoded into neural networks in order to adjudicate conflicts between rules.