Karlsruhe Institute of Technology

Improving a Hierarchical Evolutionary Algorithm with Applications in Optimization and Machine Learning

Evolutionary Graph Generation

Michael Neumann (mneumann@ntecs.de)

Institute of Theoretical Informatics

Topic of my Thesis

- Graph Generator based on EA

Topic of my Thesis

- Graph Generator based on EA
- Suitable for complex graphs

Topic of my Thesis

- Graph Generator based on EA
- Suitable for complex graphs
  - Graphs which exhibit structural similarities

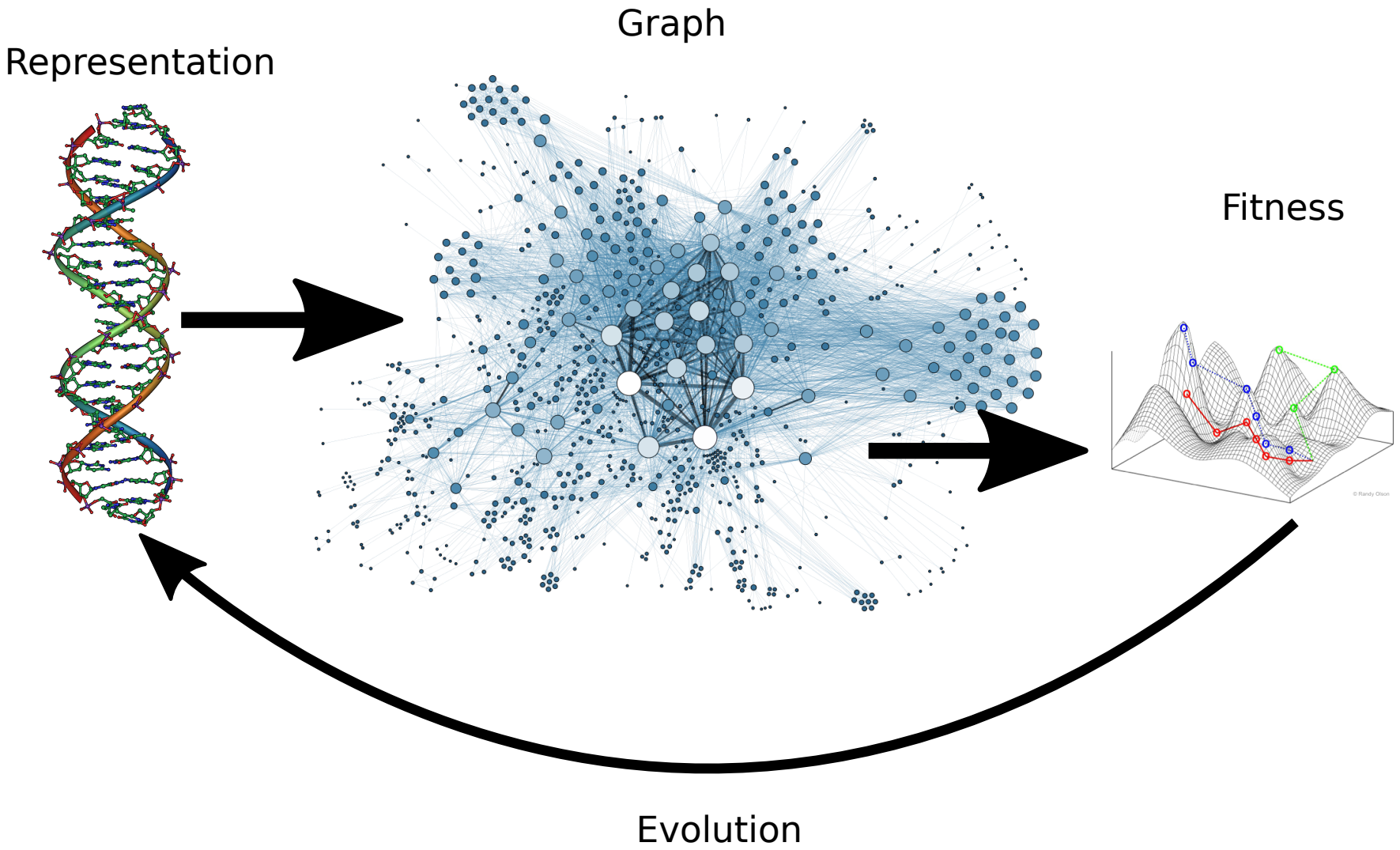
Topic of my Thesis

- Graph Generator based on EA
- Suitable for complex graphs
  - Graphs which exhibit structural similarities
- How to efficiently represent repetitive patterns?

Topic of my Thesis

- Graph Generator based on EA
- Suitable for complex graphs
  - Graphs which exhibit structural similarities
- How to efficiently represent repetitive patterns?
  - Suitable for evolution

Evolutionary Graph Generation



Topics

- Introduction
- Graph Encodings
- Our Approach
- Results and Demo
- Conclusion

Graph Encodings

Requirements

- Efficient encoding of *repetitive* patterns

Requirements

- Efficient encoding of *repetitive* patterns
- Encodes *topology* **and** connection *weights*

Requirements

- Efficient encoding of *repetitive* patterns
- Encodes *topology* **and** connection *weights*
- Suitable for evolutionary optimization (effective *genetic recombination*)

Adjacency Matrix

- Direct encoding
- Fixed length (number of nodes fixed)
- No efficient encoding of repetitive patterns

Edge List

- Direct encoding
- Variable length
- Multiple connection between pairs of nodes
- No efficient encoding of repetitive patterns

Grammar Encoding (Kitano 1990)

- First documented indirect encoding method

Grammar Encoding (Kitano 1990)

- First documented indirect encoding method
- Encodes binary connection matrix using L-system

Grammar Encoding (Kitano 1990)

- First documented indirect encoding method
- Encodes binary connection matrix using L-system
- Uses 2x2 matrices

Grammar Encoding (Kitano 1990)

$$S \to \begin{pmatrix} A & B \\ C & D \end{pmatrix}, A \to \begin{pmatrix} a & a \\ a & a \end{pmatrix}, B \to \begin{pmatrix} i & i \\ i & a \end{pmatrix}, C \to \begin{pmatrix} i & a \\ a & c \end{pmatrix}, D \to \begin{pmatrix} a & e \\ a & e \end{pmatrix}$$

$$a \to \begin{pmatrix} 0 & 0 \\ 0 & 0 \end{pmatrix}, c \to \begin{pmatrix} 0 & 0 \\ 1 & 0 \end{pmatrix}, e \to \begin{pmatrix} 0 & 1 \\ 0 & 0 \end{pmatrix}, i \to \begin{pmatrix} 1 & 0 \\ 0 & 0 \end{pmatrix}$$

Grammar Encoding (Kitano 1990)

$$S \to \begin{pmatrix} A & B \\ C & D \end{pmatrix}, A \to \begin{pmatrix} a & a \\ a & a \end{pmatrix}, B \to \begin{pmatrix} i & i \\ i & a \end{pmatrix}, C \to \begin{pmatrix} i & a \\ a & c \end{pmatrix}, D \to \begin{pmatrix} a & e \\ a & e \end{pmatrix}$$

$$a \to \begin{pmatrix} 0 & 0 \\ 0 & 0 \end{pmatrix}, c \to \begin{pmatrix} 0 & 0 \\ 1 & 0 \end{pmatrix}, e \to \begin{pmatrix} 0 & 1 \\ 0 & 0 \end{pmatrix}, i \to \begin{pmatrix} 1 & 0 \\ 0 & 0 \end{pmatrix}$$

S

Grammar Encoding (Kitano 1990)

$$S \to \begin{pmatrix} A & B \\ C & D \end{pmatrix}, A \to \begin{pmatrix} a & a \\ a & a \end{pmatrix}, B \to \begin{pmatrix} i & i \\ i & a \end{pmatrix}, C \to \begin{pmatrix} i & a \\ a & c \end{pmatrix}, D \to \begin{pmatrix} a & e \\ a & e \end{pmatrix}$$

$$a \to \begin{pmatrix} 0 & 0 \\ 0 & 0 \end{pmatrix}, c \to \begin{pmatrix} 0 & 0 \\ 1 & 0 \end{pmatrix}, e \to \begin{pmatrix} 0 & 1 \\ 0 & 0 \end{pmatrix}, i \to \begin{pmatrix} 1 & 0 \\ 0 & 0 \end{pmatrix}$$

$$S \to \begin{pmatrix} A & B \\ C & D \end{pmatrix}$$

Grammar Encoding (Kitano 1990)

$$S \to \begin{pmatrix} A & B \\ C & D \end{pmatrix}, A \to \begin{pmatrix} a & a \\ a & a \end{pmatrix}, B \to \begin{pmatrix} i & i \\ i & a \end{pmatrix}, C \to \begin{pmatrix} i & a \\ a & c \end{pmatrix}, D \to \begin{pmatrix} a & e \\ a & e \end{pmatrix}$$

$$a \to \begin{pmatrix} 0 & 0 \\ 0 & 0 \end{pmatrix}, c \to \begin{pmatrix} 0 & 0 \\ 1 & 0 \end{pmatrix}, e \to \begin{pmatrix} 0 & 1 \\ 0 & 0 \end{pmatrix}, i \to \begin{pmatrix} 1 & 0 \\ 0 & 0 \end{pmatrix}$$

$$S \to \begin{pmatrix} A & B \\ C & D \end{pmatrix} \to \begin{pmatrix} a & a & i & i \\ a & a & i & a \\ i & a & a & e \\ a & c & a & e \end{pmatrix}$$

Grammar Encoding (Kitano 1990)

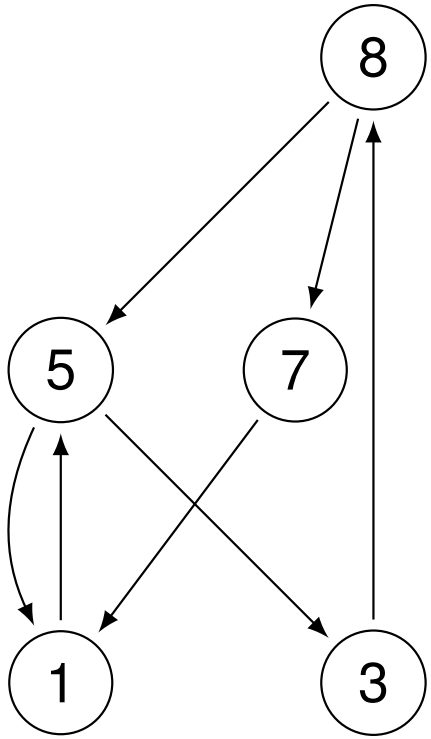
$$S \to \begin{pmatrix} A & B \\ C & D \end{pmatrix}, A \to \begin{pmatrix} a & a \\ a & a \end{pmatrix}, B \to \begin{pmatrix} i & i \\ i & a \end{pmatrix}, C \to \begin{pmatrix} i & a \\ a & c \end{pmatrix}, D \to \begin{pmatrix} a & e \\ a & e \end{pmatrix}$$

$$a \to \begin{pmatrix} 0 & 0 \\ 0 & 0 \end{pmatrix}, c \to \begin{pmatrix} 0 & 0 \\ 1 & 0 \end{pmatrix}, e \to \begin{pmatrix} 0 & 1 \\ 0 & 0 \end{pmatrix}, i \to \begin{pmatrix} 1 & 0 \\ 0 & 0 \end{pmatrix}$$

$$S \to \begin{pmatrix} A & B \\ C & D \end{pmatrix} \to \begin{pmatrix} a & a & i & i \\ a & a & i & a \\ i & a & a & e \\ a & c & a & e \end{pmatrix} \to \begin{pmatrix} 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \end{pmatrix}$$

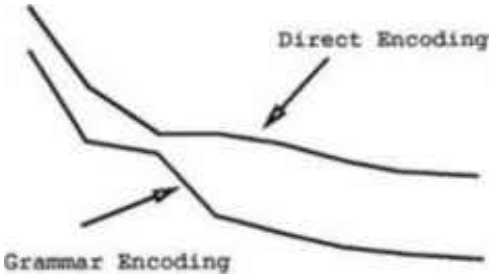
Grammar Encoding (Kitano 1990)

$$\begin{pmatrix} 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \end{pmatrix}$$



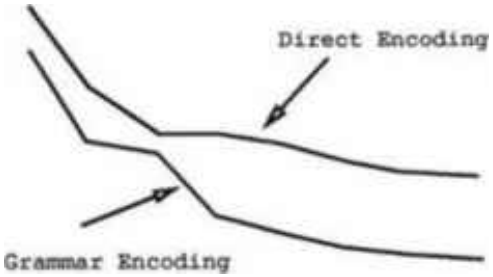
Grammar Encoding (Kitano 1990)

- Shows significantly higher scalability and faster convergence



Grammar Encoding (Kitano 1990)

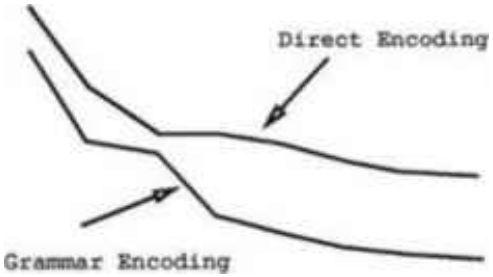
- Shows significantly higher scalability and faster convergence



- Only binary matrix

Grammar Encoding (Kitano 1990)

- Shows significantly higher scalability and faster convergence



- Only binary matrix
- Matrix size power of 2

Cellular Encoding (Gruau 1992)

- Evolves a graph from a single cell (= node)

Cellular Encoding (Gruau 1992)

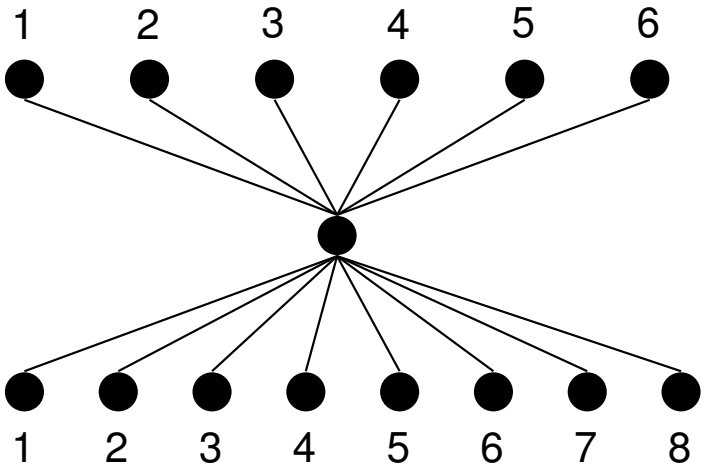
- Evolves a graph from a single cell (= node)
- Based on the concept of *cell division*

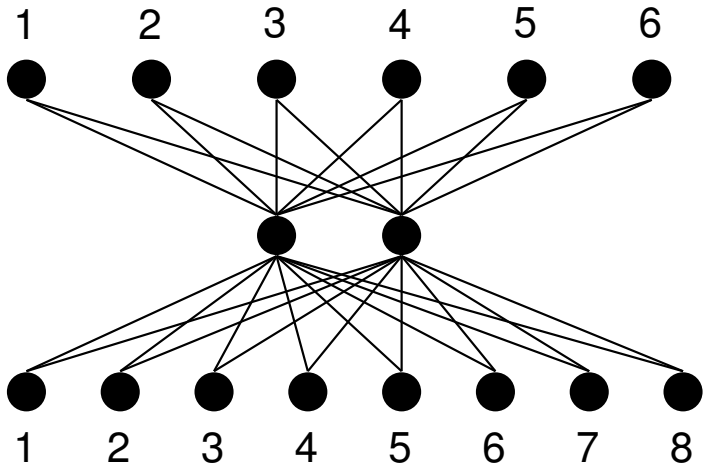
Cellular Encoding (Gruau 1992)

- Evolves a graph from a single cell (= node)
- Based on the concept of *cell division*
- Graph transformations are node-centric. Applied in parallel

Cellular Encoding (Gruau 1992)

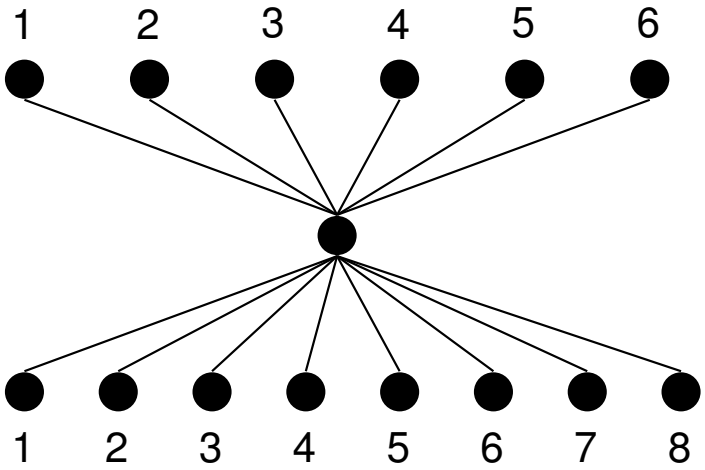
- Parallel Division (P)

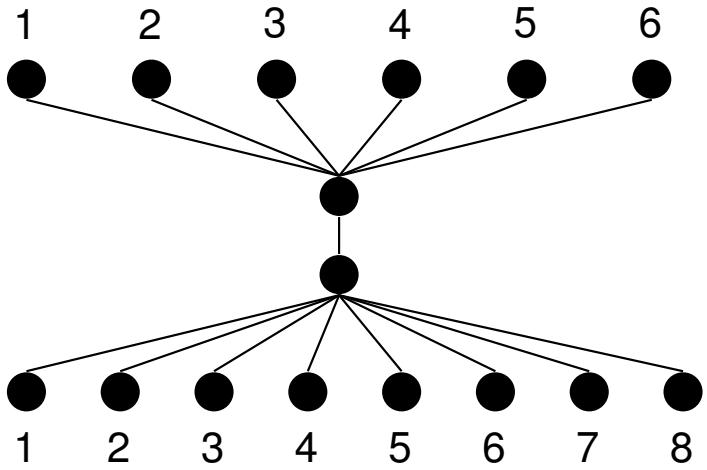




Cellular Encoding (Gruau 1992)

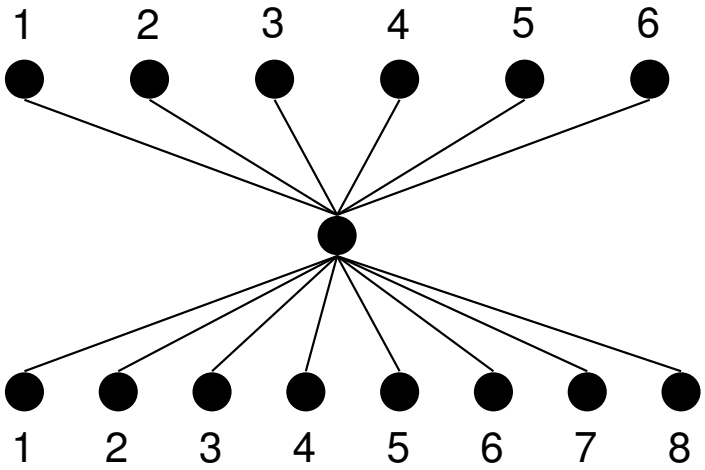
- Sequential Division (S)

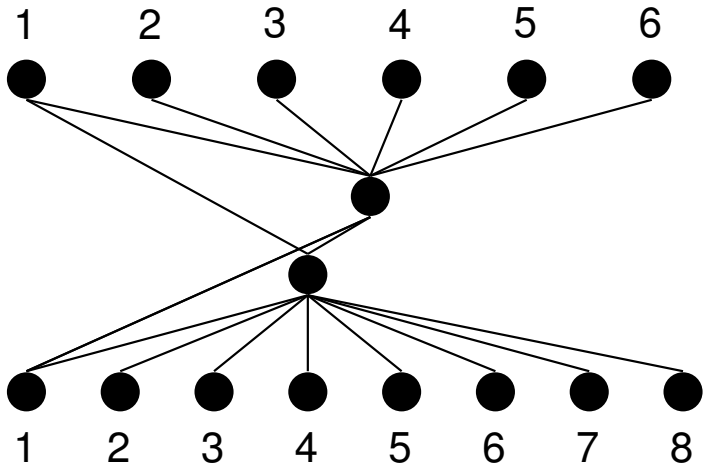




Cellular Encoding (Gruau 1992)

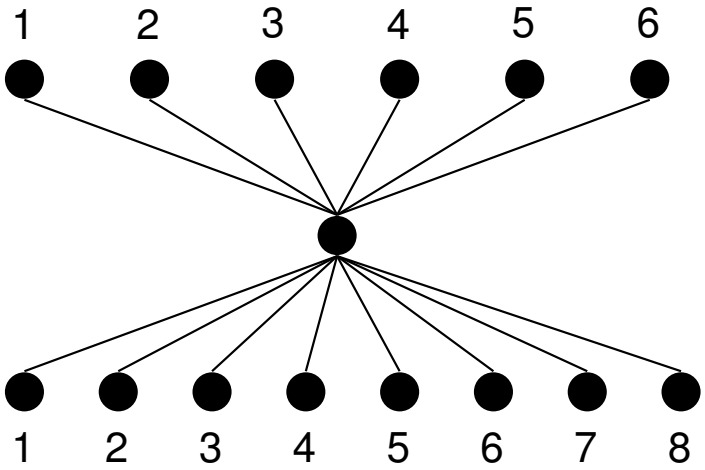
- Division `T`, variation of `S`

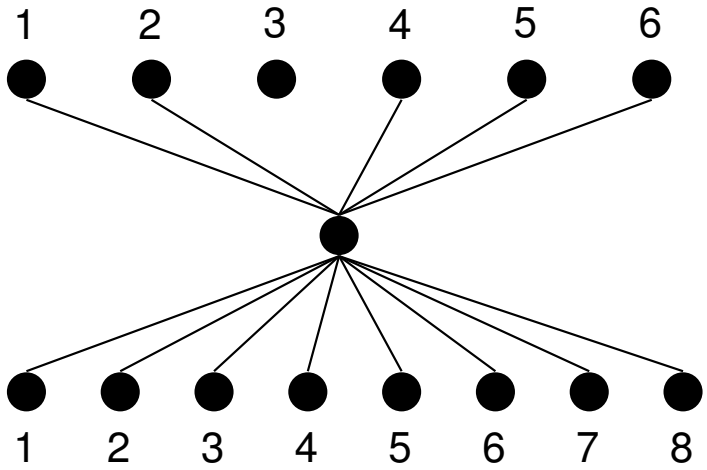




Cellular Encoding (Gruau 1992)

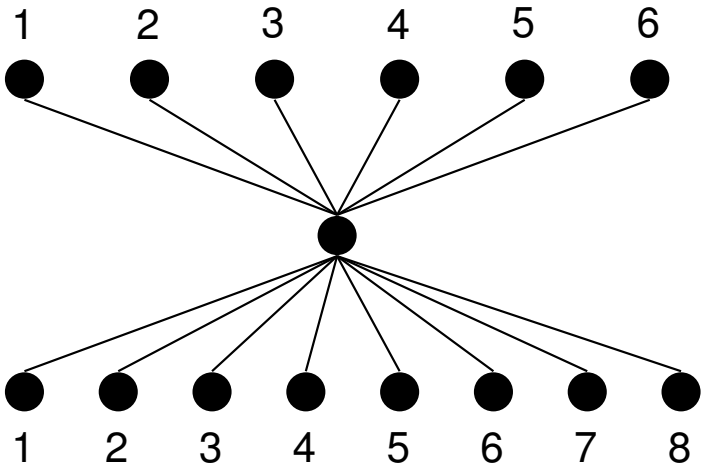
- Disconnect C_3

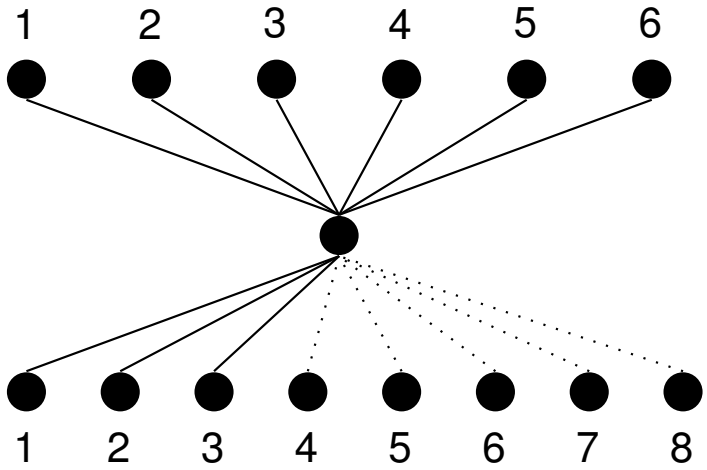




Cellular Encoding (Gruau 1992)

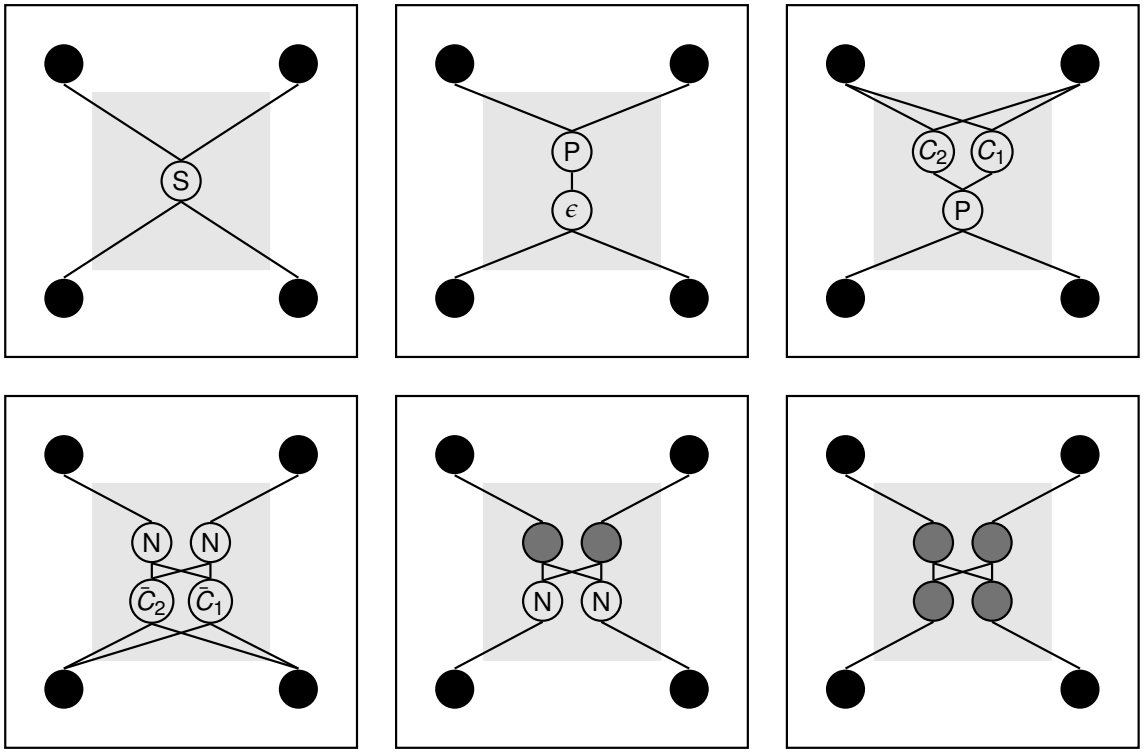
- Modify weight K_4

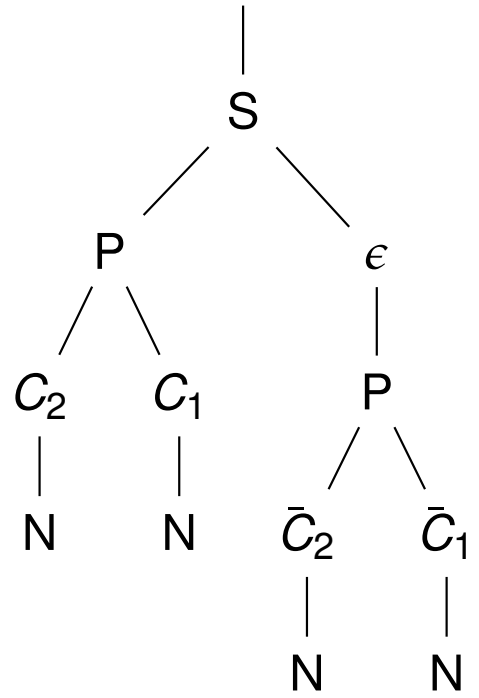




Cellular Encoding (Gruau 1992)

- Starts with zygote cell embedded in environment
- Instructions encoded in a binary tree





Cellular Encoding (Gruau 1992)

- Very powerful graph transformations

Cellular Encoding (Gruau 1992)

- Very powerful graph transformations
- Explicit specification of connections

Cellular Encoding (Gruau 1992)

- Very powerful graph transformations
- Explicit specification of connections
- No efficient genetic recombination (no gene alignment)

Cellular Encoding (Gruau 1992)

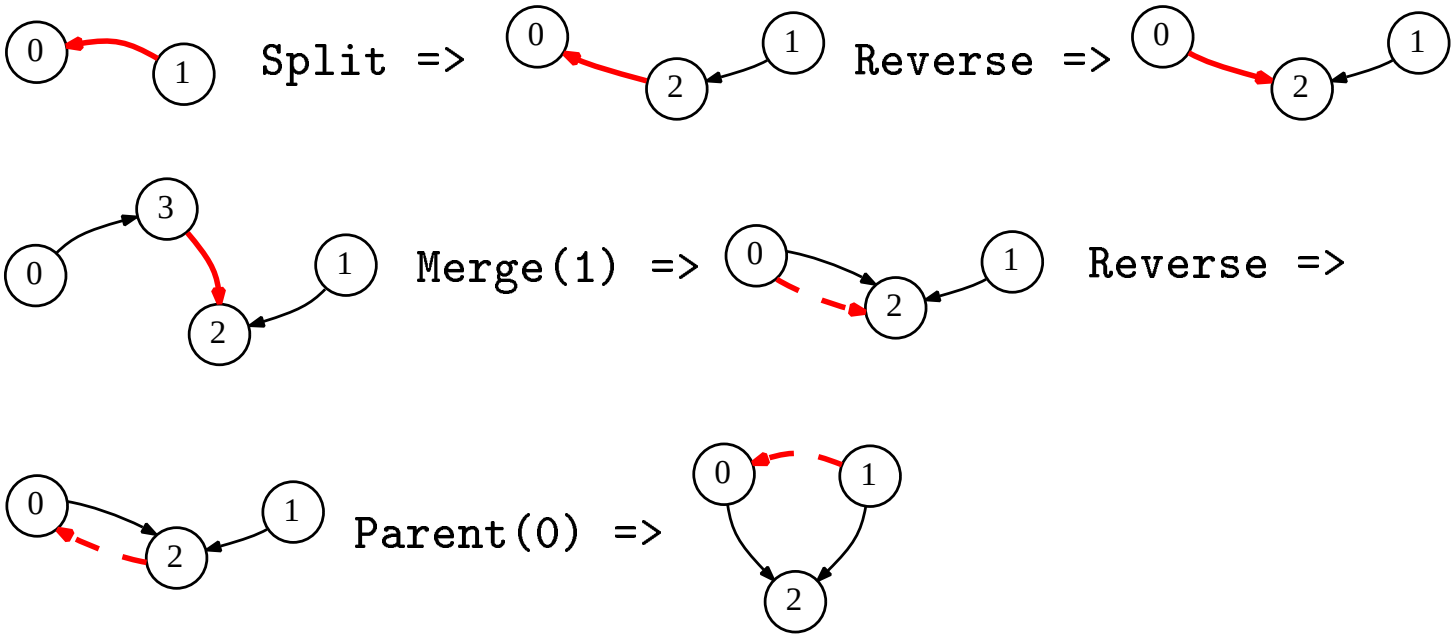
- Very powerful graph transformations
- Explicit specification of connections
- No efficient genetic recombination (no gene alignment)
- Weight patterns (gradients) cannot be encoded efficiently

L-System (Hornby and Pollack 2001)

- Parametric L-System to generate list of *edge* transformations

L-System (Hornby and Pollack 2001)

- Transformations applied *sequentially* on active edge



L-System (Hornby and Pollack 2001)

- Definition of parametric modules

L-System (Hornby and Pollack 2001)

- Definition of parametric modules
- Needs many transformations even for simple graphs

L-System (Hornby and Pollack 2001)

- Definition of parametric modules
- Needs many transformations even for simple graphs
- No efficient genetic recombination (no gene alignment)

Hierarchical Encoding (Schmidt 2009)

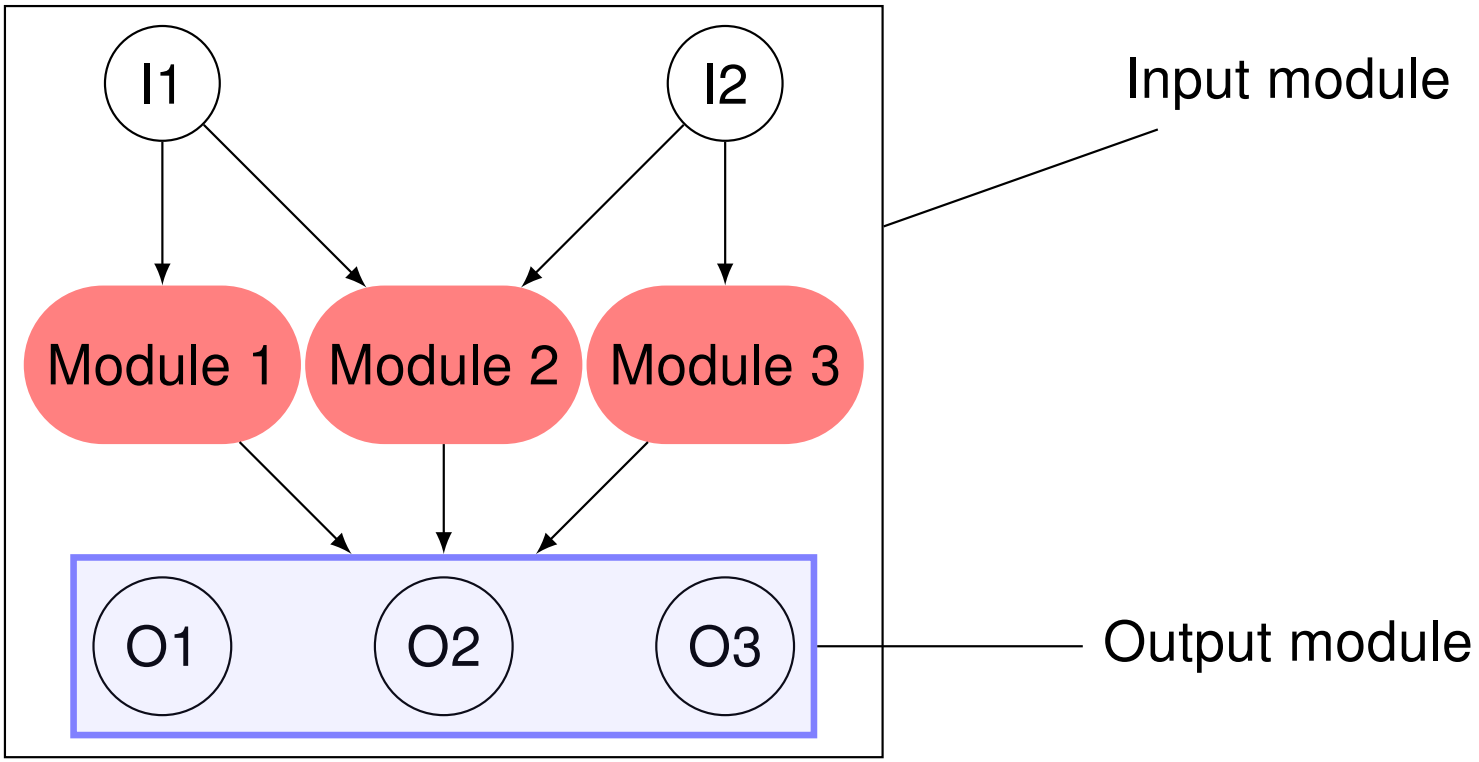
- Genome contains module definitions which can be instanciated

Hierarchical Encoding (Schmidt 2009)

- Genome contains module definitions which can be instanciated

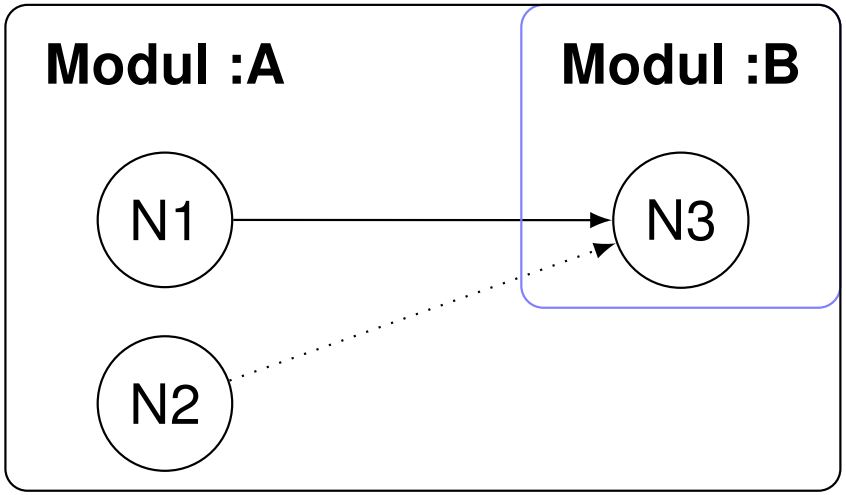
Module def 1	Module def 2	Module def 3

- Modules can instanciate submodules recursively



Hierarchical Encoding (Schmidt 2009)

A: **nanacx**	B: **nb**



Problems of Hierarchical Encoding

- Realization of intermodular connections

Problems of Hierarchical Encoding

- Realization of intermodular connections
- Inefficient recombination

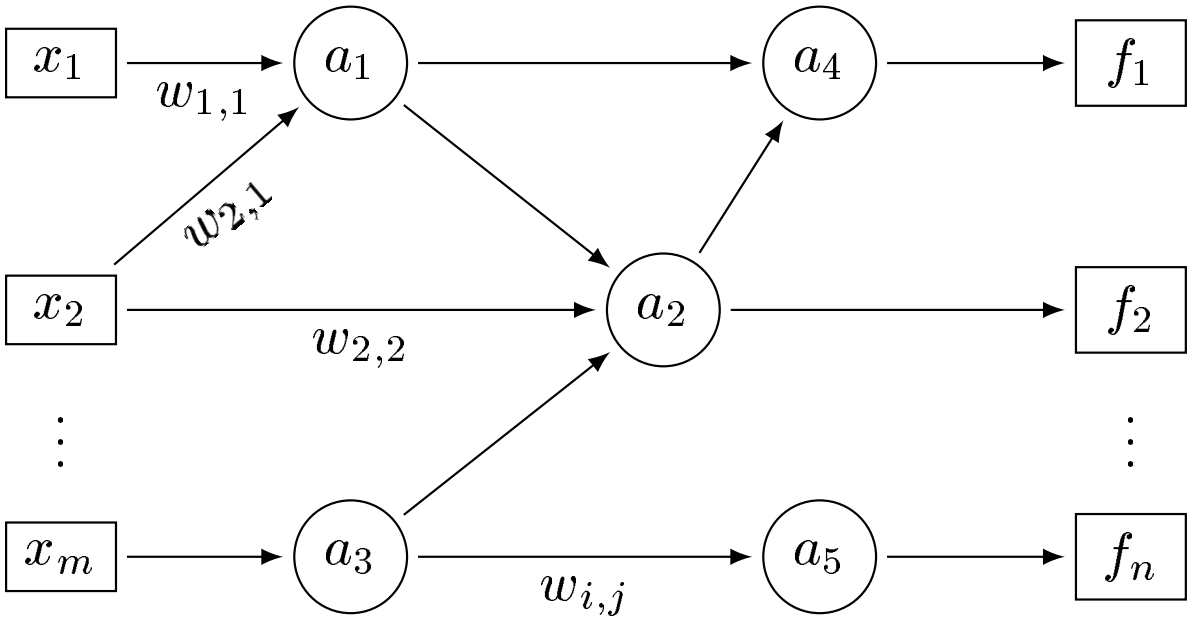
Problems of Hierarchical Encoding

- Realization of intermodular connections
- Inefficient recombination
- Module variation only by copying a whole module definition

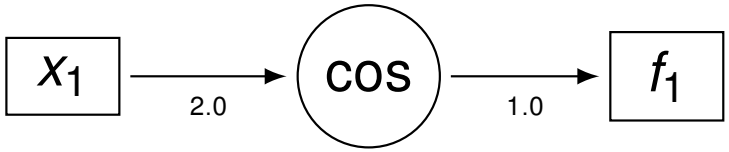
Our Approach

Compositional Pattern Producing Networks

- weighted, acyclic, directed network
- inputs $x_i \in \mathcal{R}$
- outputs $f_i : (x_1, \ldots, x_m) \rightarrow \mathcal{R}$
- activation functions $a_i \in \{\sin, \mathit{atan}, \mathit{gauss}, \mathit{id}, \mathit{const}, \ldots\}$
- accumulation functions Σ, Π

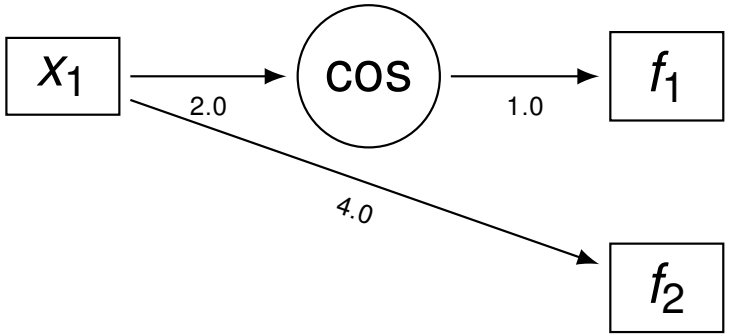


Compositional Pattern Producing Networks



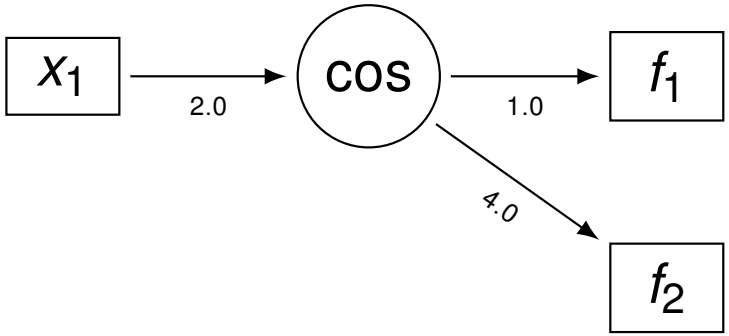
- $f_1 = 1 \cdot \cos(2 \cdot x_1)$

Compositional Pattern Producing Networks



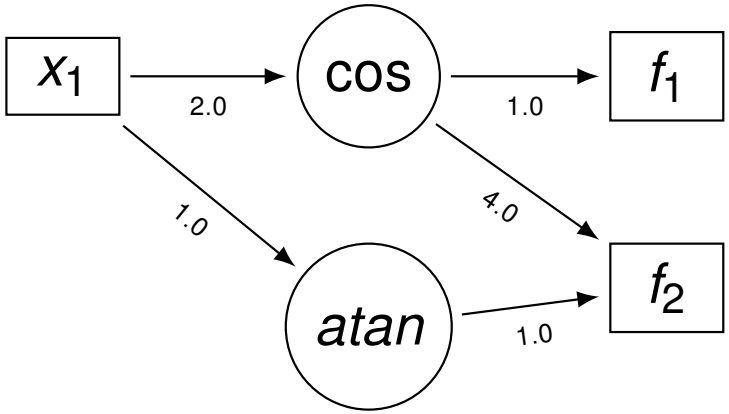
- $f_1 = 1 \cdot \cos(2 \cdot x_1)$
- $f_2 = 4 \cdot x_1$

Compositional Pattern Producing Networks



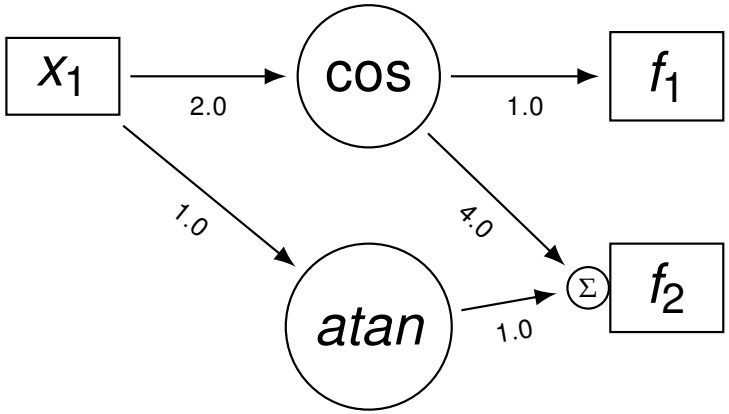
- $f_1 = 1 \cdot \cos(2 \cdot x_1)$
- $f_2 = 4 \cdot \cos(2 \cdot x_1)$

Compositional Pattern Producing Networks



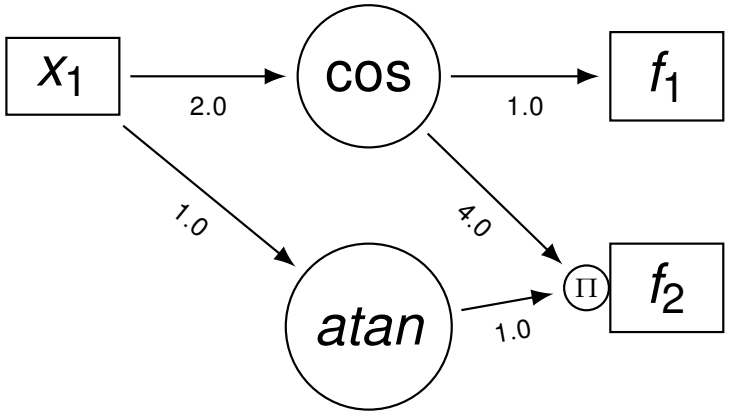
- $f_1 = 1 \cdot \cos(2 \cdot x_1)$
- $f_2 = (4 \cdot \cos(2 \cdot x_1)) \circ atan(x_1)$

Compositional Pattern Producing Networks



- $f_1 = 1 \cdot \cos(2 \cdot x_1)$
- $f_2 = (4 \cdot \cos(2 \cdot x_1)) + atan(x_1)$

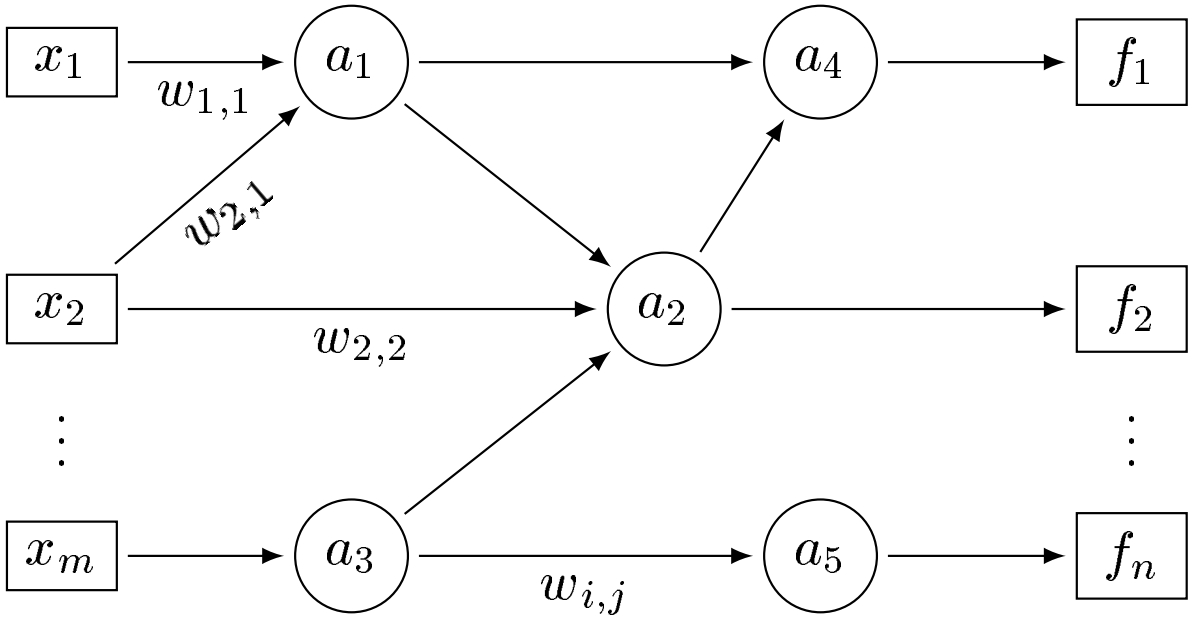
Compositional Pattern Producing Networks



- $f_1 = 1 \cdot \cos(2 \cdot x_1)$
- $f_2 = (4 \cdot \cos(2 \cdot x_1)) \cdot atan(x_1)$

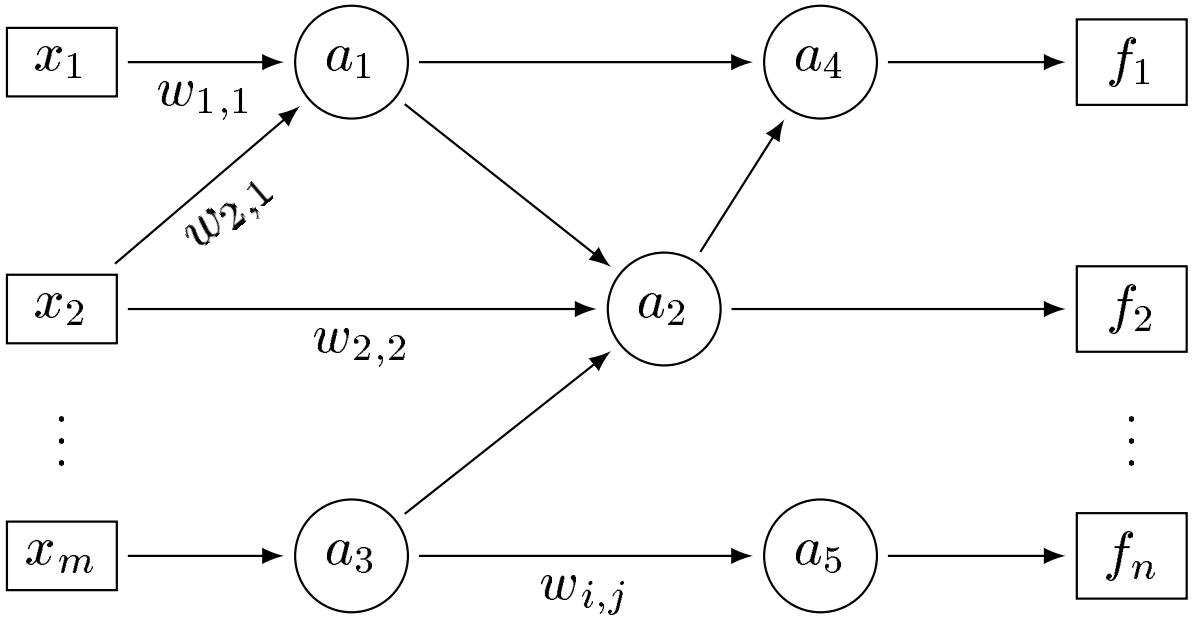
Compositional Pattern Producing Networks

- CPPNs represent a set of functions $f_i : (x_1, \ldots, x_m) \rightarrow \mathcal{R}$



Compositional Pattern Producing Networks

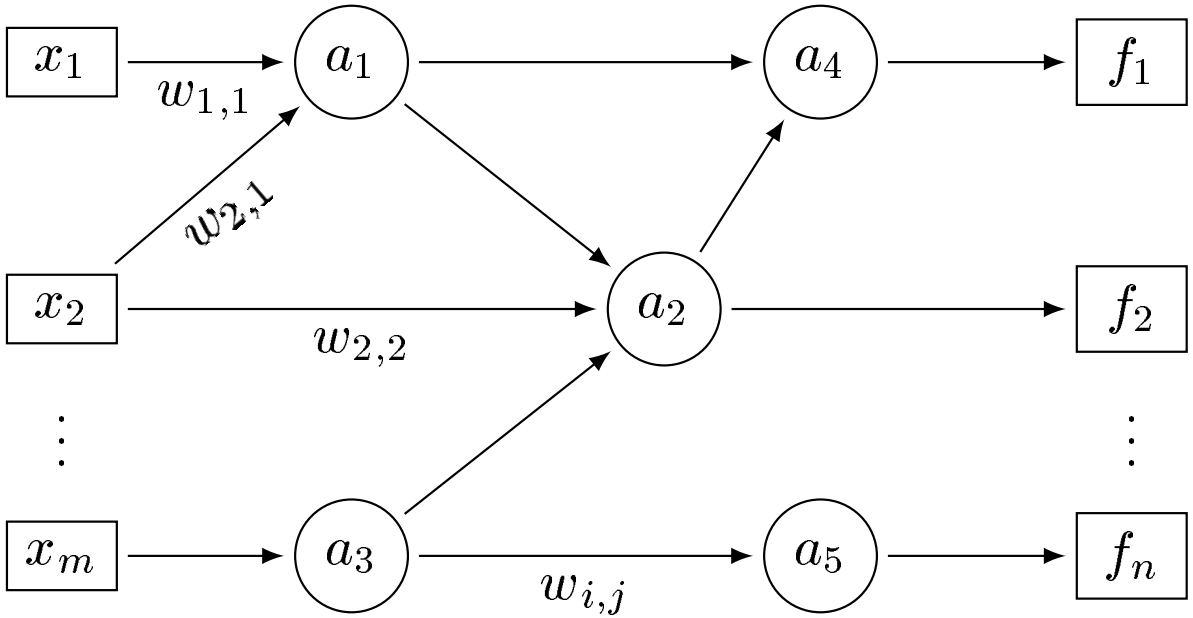
- CPPNs represent a set of functions $f_i : (x_1, \ldots, x_m) \rightarrow \mathcal{R}$



- Function terms can be shared

Compositional Pattern Producing Networks

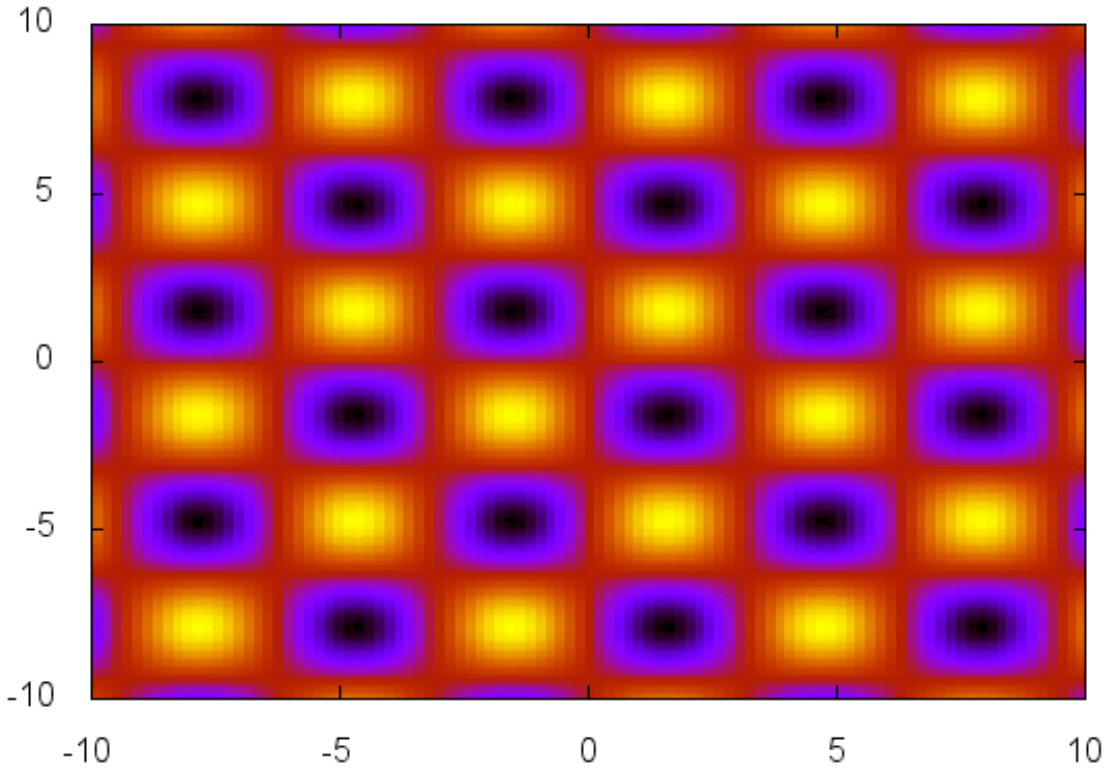
- CPPNs represent a set of functions $f_i : (x_1, \ldots, x_m) \rightarrow \mathcal{R}$



- Function terms can be shared
- Graph representation allows use of topological analysis

Compositional Pattern Producing Networks

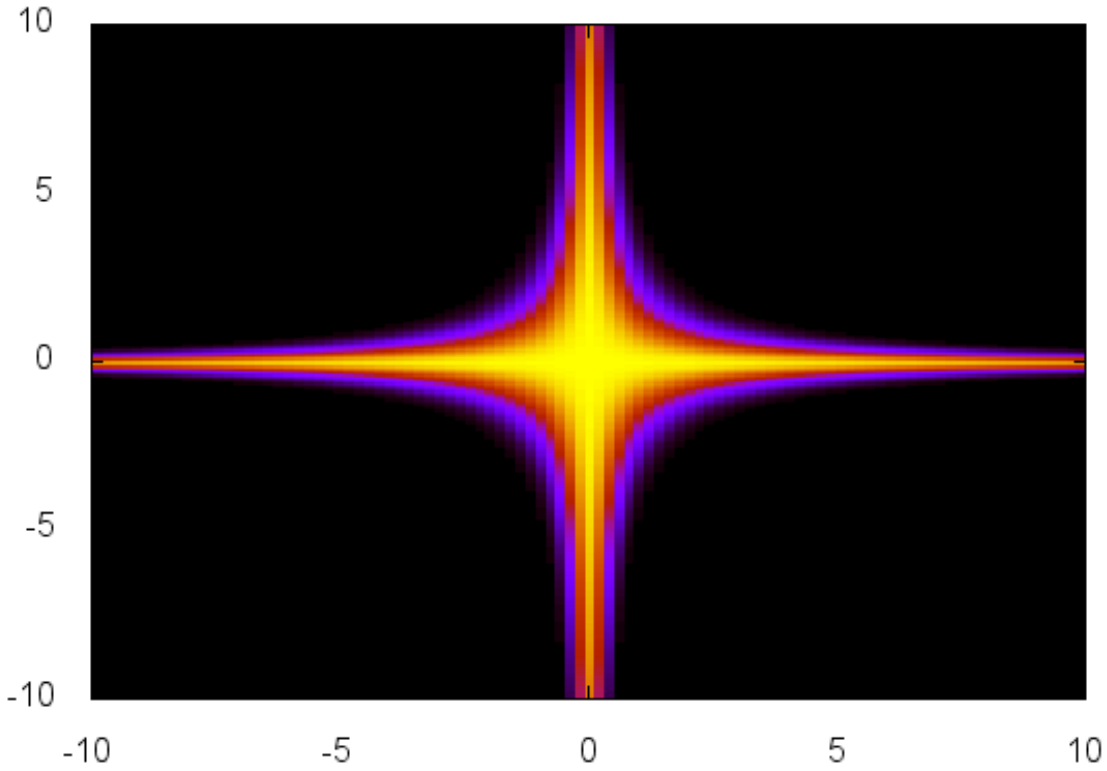
- Periodic functions (sin): Repetition



$$sin(x) * sin(y)$$

Compositional Pattern Producing Networks

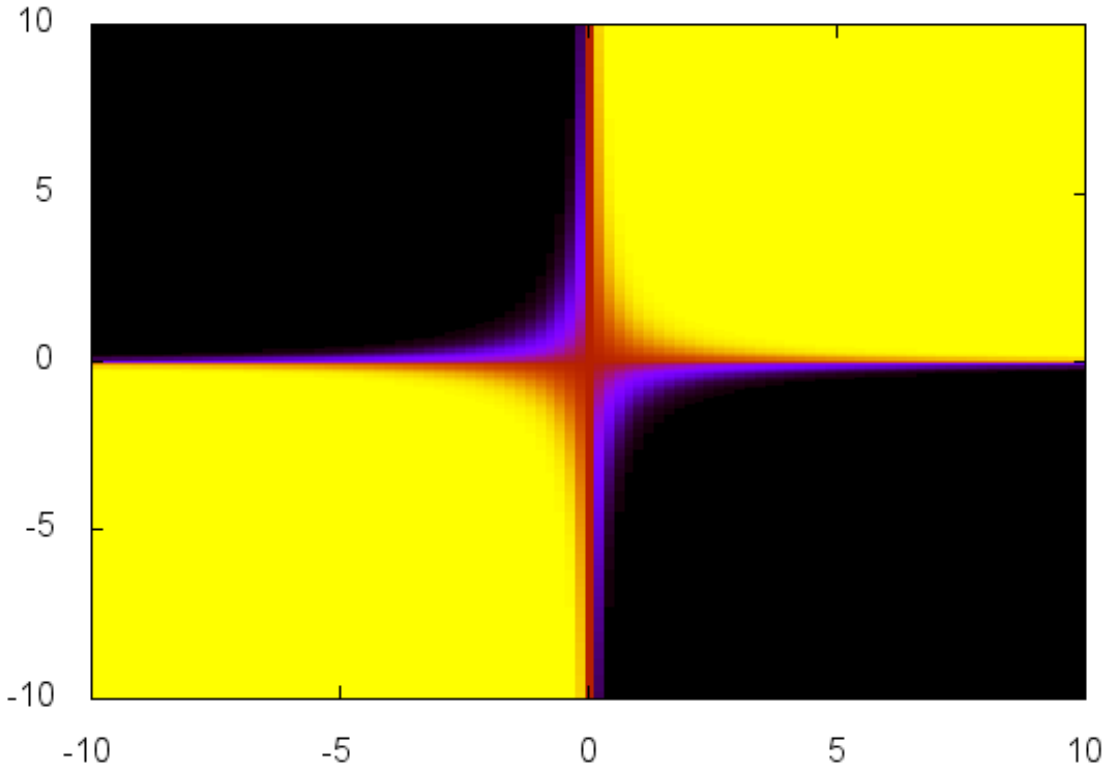
- Symmetric functions (Gaussian)



$$G(x * y)$$

Compositional Pattern Producing Networks

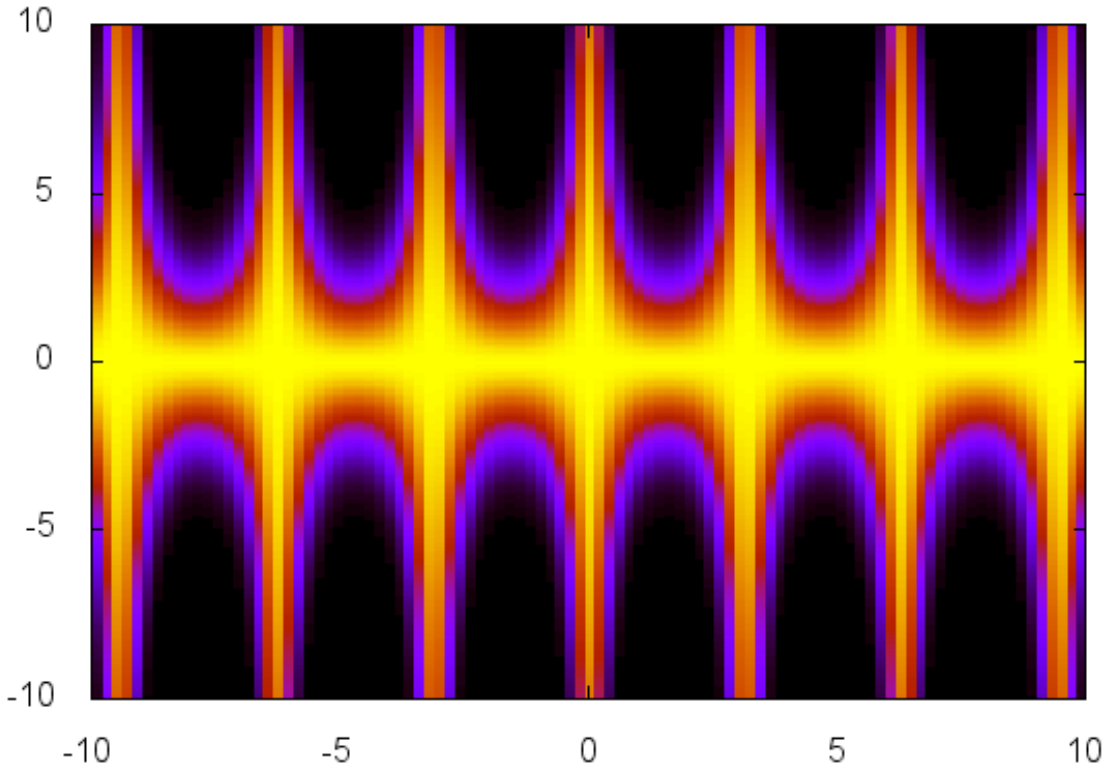
- Imperfect symmetric function (arctan)



$$atan(x * y)$$

Compositional Pattern Producing Networks

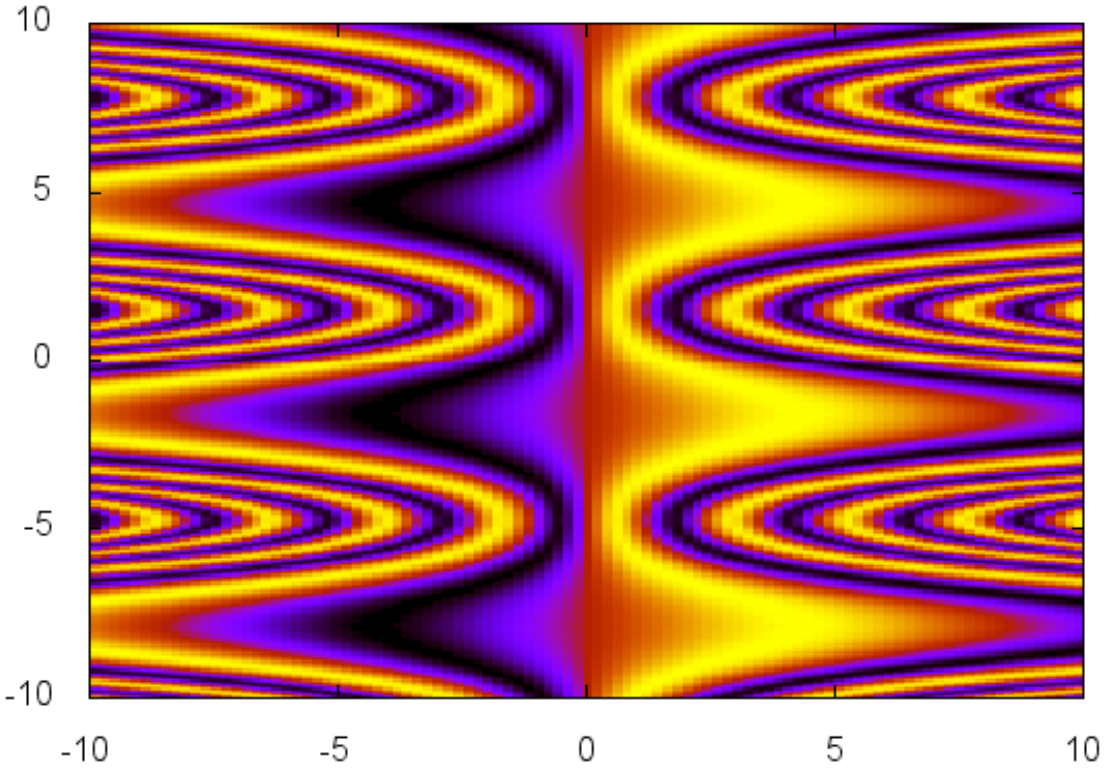
- Combination of periodic and symmetric



$$G(sin(x) * y)$$

Compositional Pattern Producing Networks

- Combination of periodic and imperfect symetric



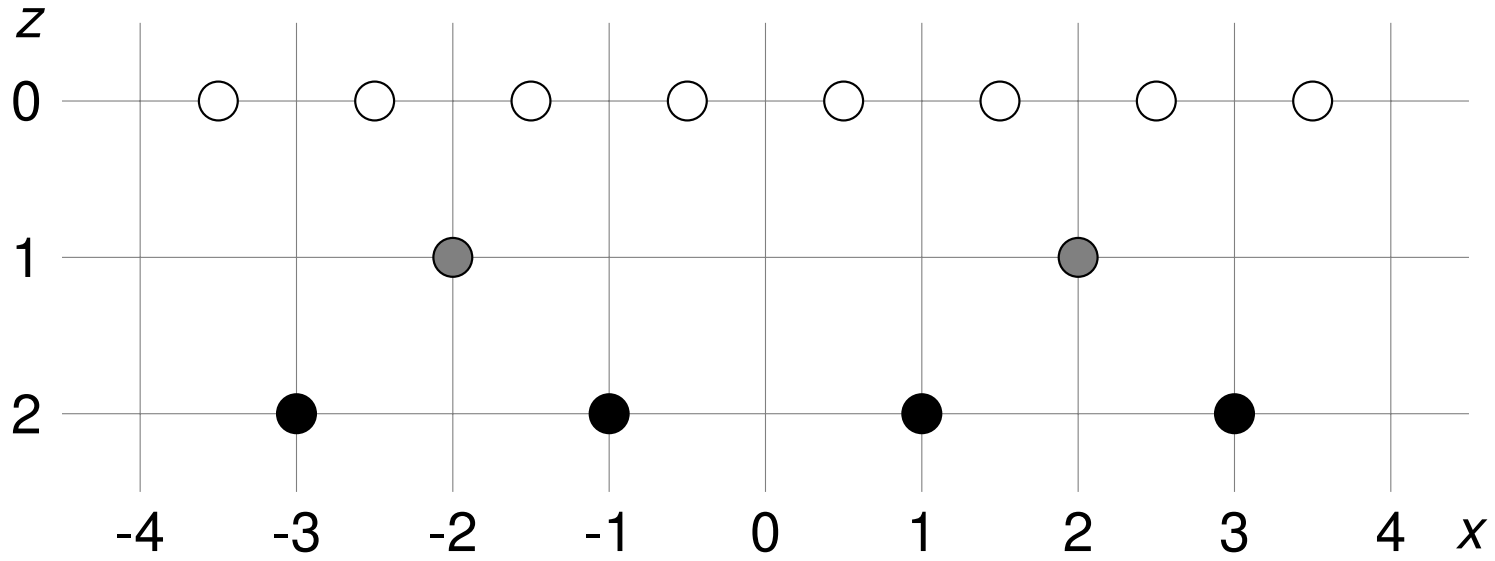
$$\sin x \cdot e^{\sin y}$$

Compositional Pattern Producing Networks

- How can we generate graphs from a CPPN?

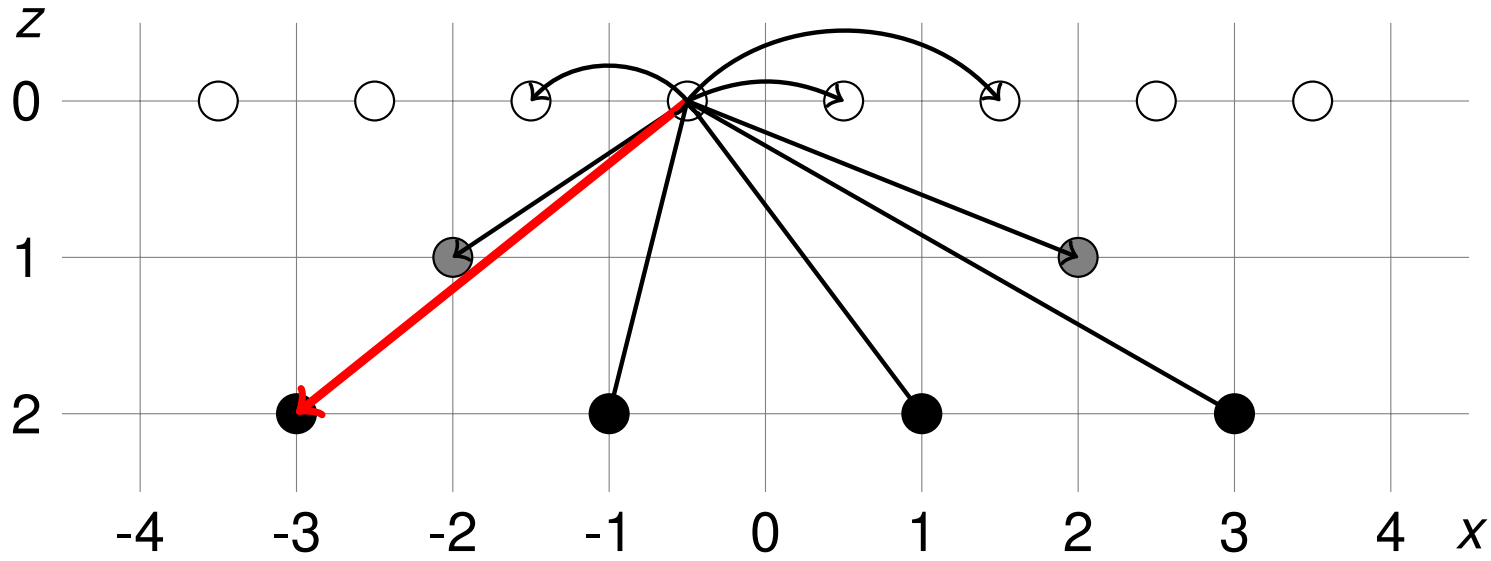
Compositional Pattern Producing Networks

- Solution: Assign coordinates to nodes



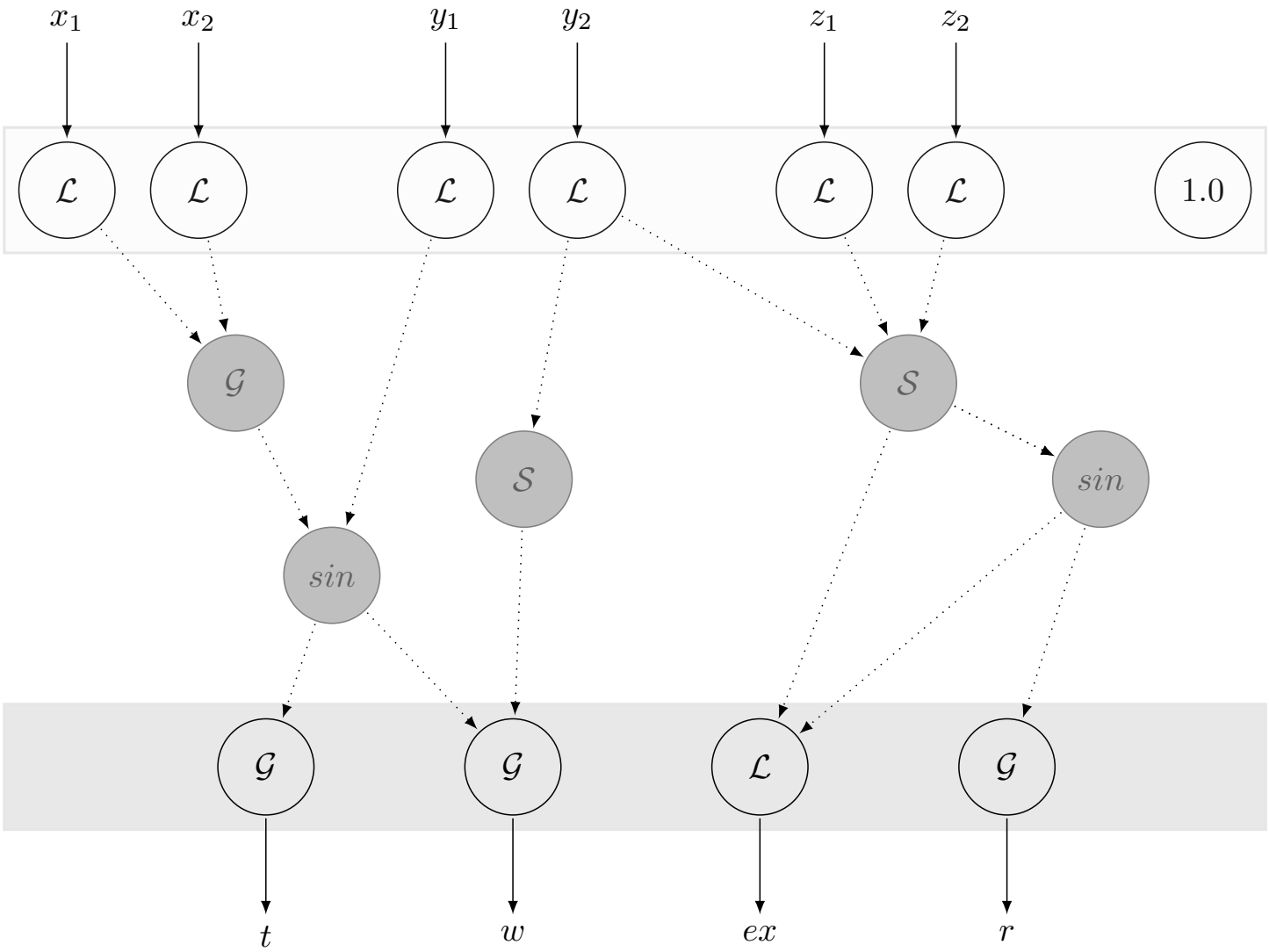
Compositional Pattern Producing Networks

- Solution: Assign coordinates to nodes



- Query CPPN for each *node pair* to determine *connectivity*

Compositional Pattern Producing Networks



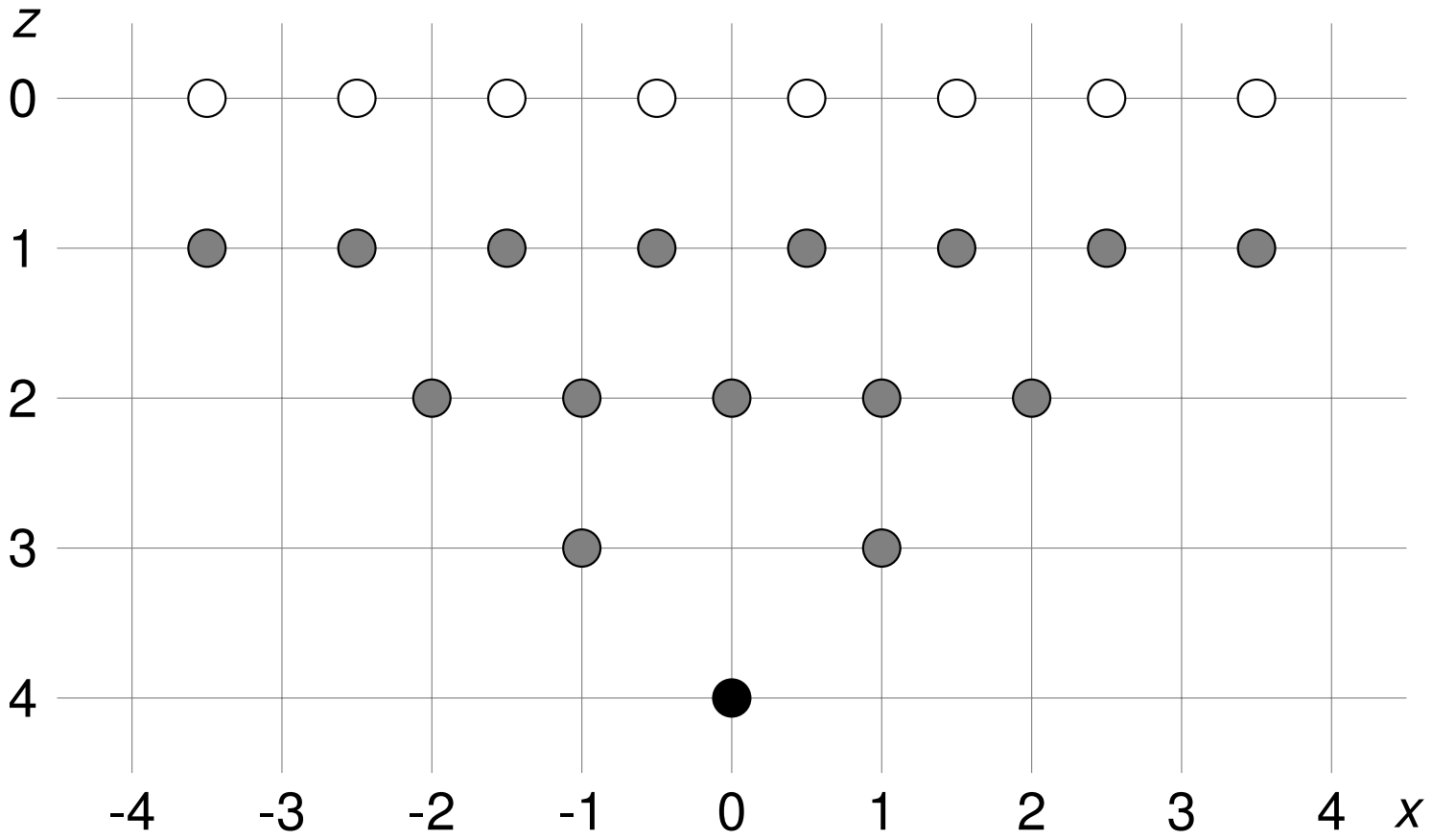
Substrate Placement

- Application specific

Substrate Placement

- Application specific
- Can make use of geometry of environmental system (sensors)

Substrate Placement



Genetic Operators

- Structural mutation (AddNode, DropNode, ModifyNode, Connect, Disconnect, ...)

Genetic Operators

- Structural mutation (AddNode, DropNode, ModifyNode, Connect, Disconnect, ...)
- Mutation of connection weights

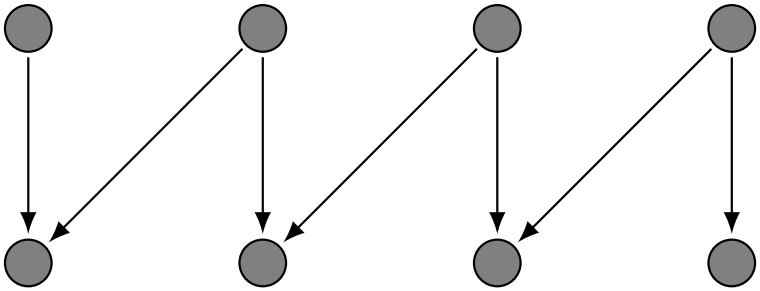
Genetic Operators

- Structural mutation (AddNode, DropNode, ModifyNode, Connect, Disconnect, ...)
- Mutation of connection weights
- Crossover of weights of matching edges

Results and Demo

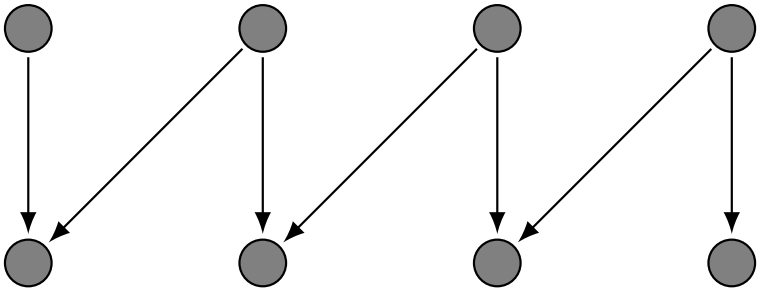
Tested Graph Motifs

- Bipartite

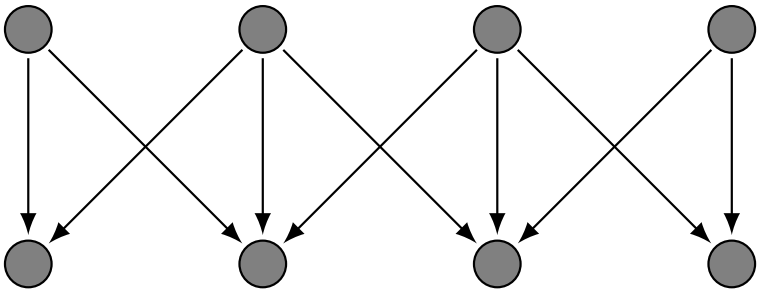


Tested Graph Motifs

- Bipartite

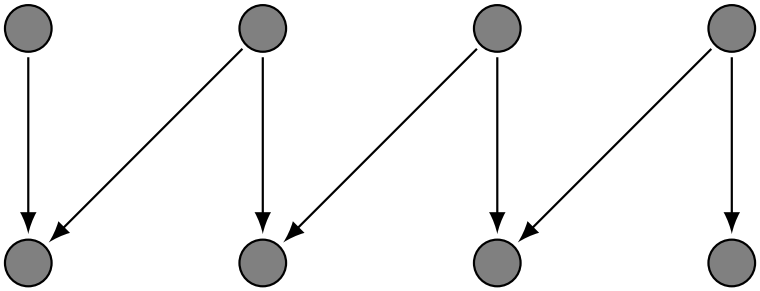


- Bi-Fan

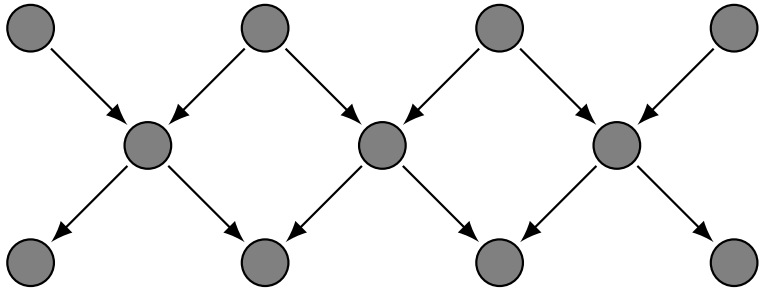


Tested Graph Motifs

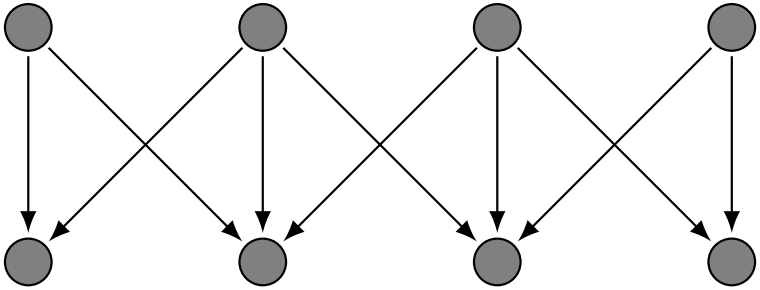
- Bipartite



- Bi-Parallel

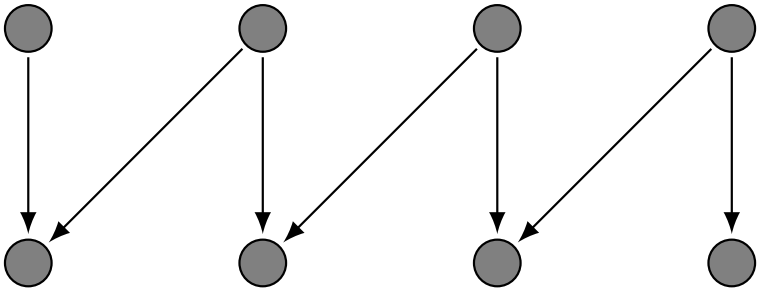


- Bi-Fan

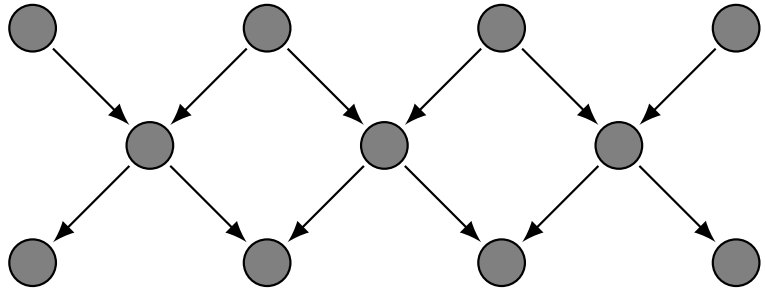


Tested Graph Motifs

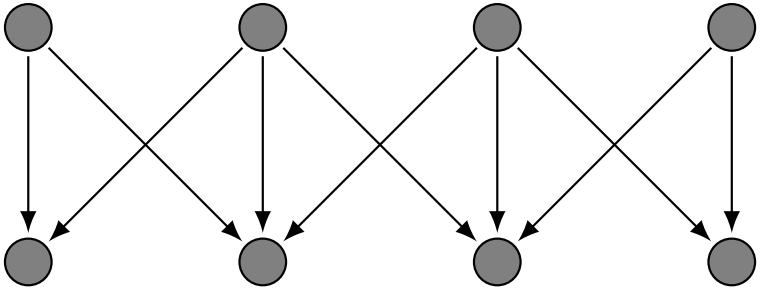
- Bipartite



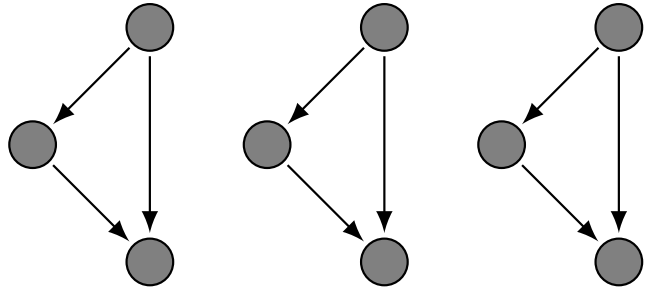
- Bi-Parallel



- Bi-Fan



- Feed-Forward Loop (FFL)



Fitness Function

- Similarity to target graph

Fitness Function

- Similarity to target graph
  - Neighbor matching (M. Nikolic 2012)
  - Triadic census

Fitness Function

- Similarity to target graph
  - Neighbor matching (M. Nikolic 2012)
  - Triadic census
- Behavioral Diversity

Fitness Function

- Similarity to target graph
  - Neighbor matching (M. Nikolic 2012)
  - Triadic census
- Behavioral Diversity
  - Compressed form of the function of the CPPN

Fitness Function

- Similarity to target graph
  - Neighbor matching (M. Nikolic 2012)
  - Triadic census
- Behavioral Diversity
  - Compressed form of the function of the CPPN
  - Average hamming distance of *behavior* to all other CPPNs of the population

Fitness Function

- Similarity to target graph
  - Neighbor matching (M. Nikolic 2012)
  - Triadic census
- Behavioral Diversity
  - Compressed form of the function of the CPPN
  - Average hamming distance of *behavior* to all other CPPNs of the population
  - Prefers CPPNs with diverse behavior (avoids domination)

Fitness Function

- Similarity to target graph
  - Neighbor matching (M. Nikolic 2012)
  - Triadic census
- Behavioral Diversity
  - Compressed form of the function of the CPPN
  - Average hamming distance of *behavior* to all other CPPNs of the population
  - Prefers CPPNs with diverse behavior (avoids domination)
- Connection Cost

Fitness Function

- Similarity to target graph
  - Neighbor matching (M. Nikolic 2012)
  - Triadic census
- Behavioral Diversity
  - Compressed form of the function of the CPPN
  - Average hamming distance of *behavior* to all other CPPNs of the population
  - Prefers CPPNs with diverse behavior (avoids domination)
- Connection Cost
  - Sum of connection lengths

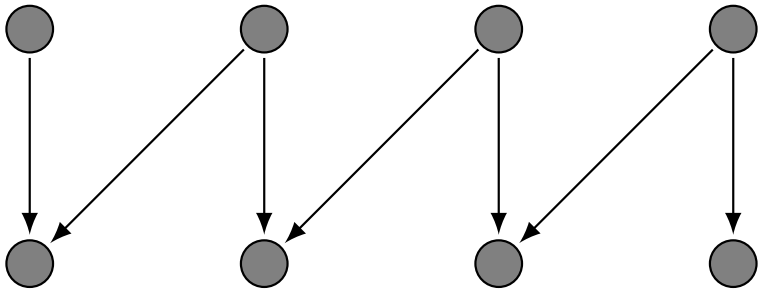
Fitness Function

- Similarity to target graph
  - Neighbor matching (M. Nikolic 2012)
  - Triadic census
- Behavioral Diversity
  - Compressed form of the function of the CPPN
  - Average hamming distance of *behavior* to all other CPPNs of the population
  - Prefers CPPNs with diverse behavior (avoids domination)
- Connection Cost
  - Sum of connection lengths
  - Prefers local connections

Fitness Function

- Similarity to target graph
  - Neighbor matching (M. Nikolic 2012)
  - Triadic census
- Behavioral Diversity
  - Compressed form of the function of the CPPN
  - Average hamming distance of *behavior* to all other CPPNs of the population
  - Prefers CPPNs with diverse behavior (avoids domination)
- Connection Cost
  - Sum of connection lengths
  - Prefers local connections
  - Results in higher modularity

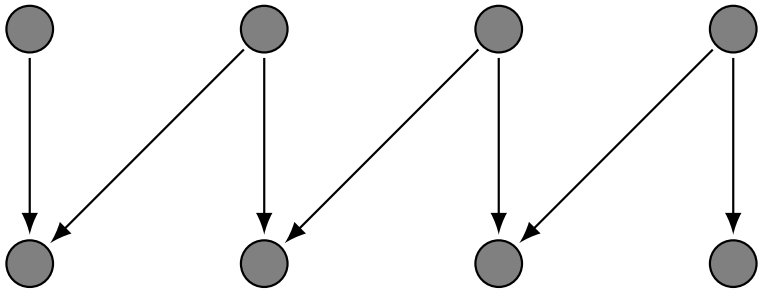
CPPNs for bipartite Graph



$z = 1$

$z = -1$

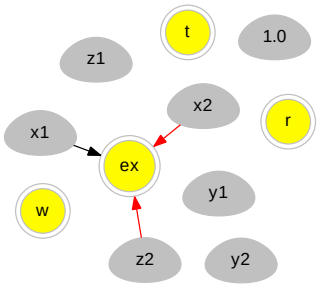
CPPNs for bipartite Graph



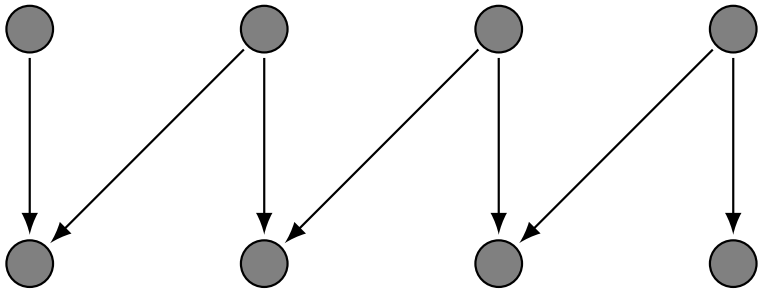
$z = 1$

$z = -1$

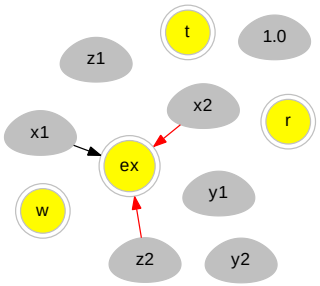
- N=5



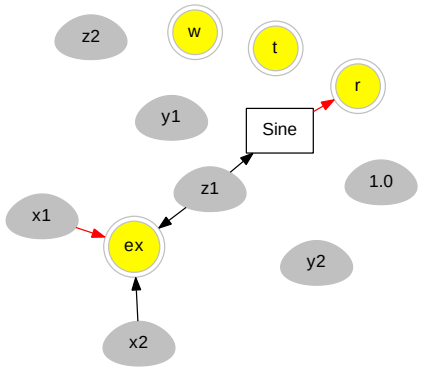
CPPNs for bipartite Graph



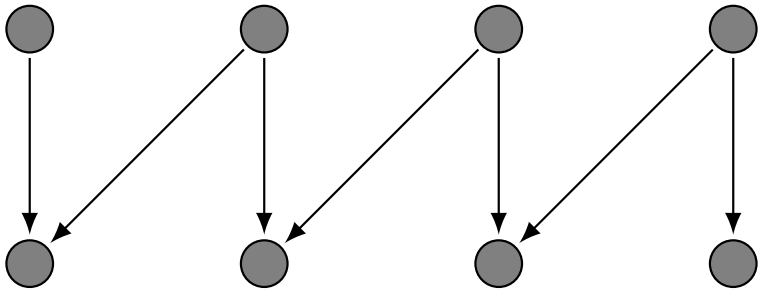
- N=5



- N=10



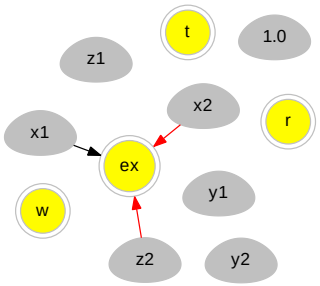
CPPNs for bipartite Graph



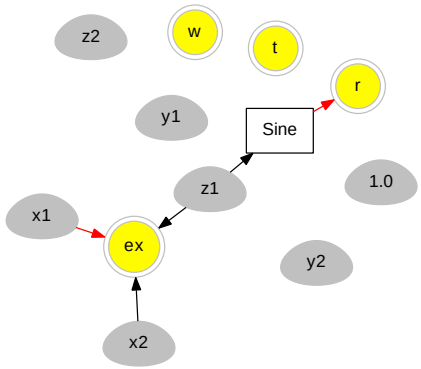
$z = 1$

$z = -1$

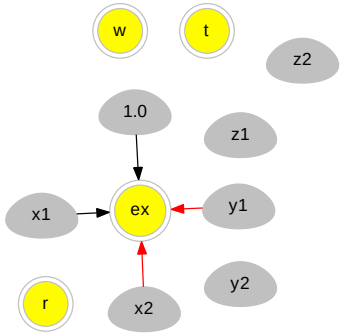
- N=5



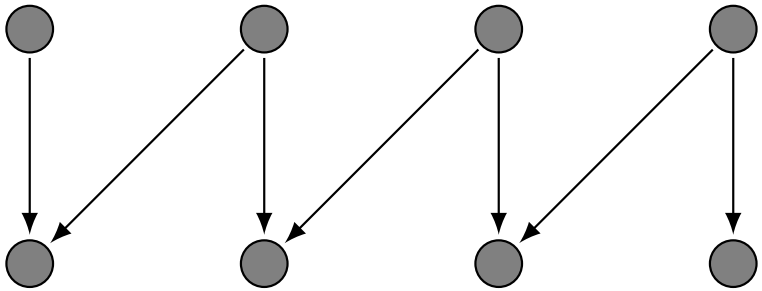
- N=10



- N=100



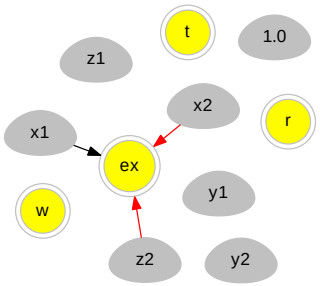
CPPNs for bipartite Graph



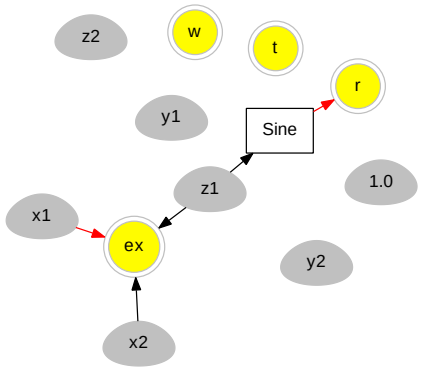
$z = 1$

$z = -1$

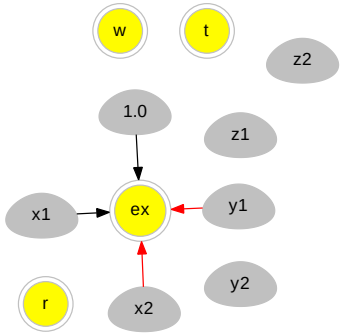
- N=5



- N=10

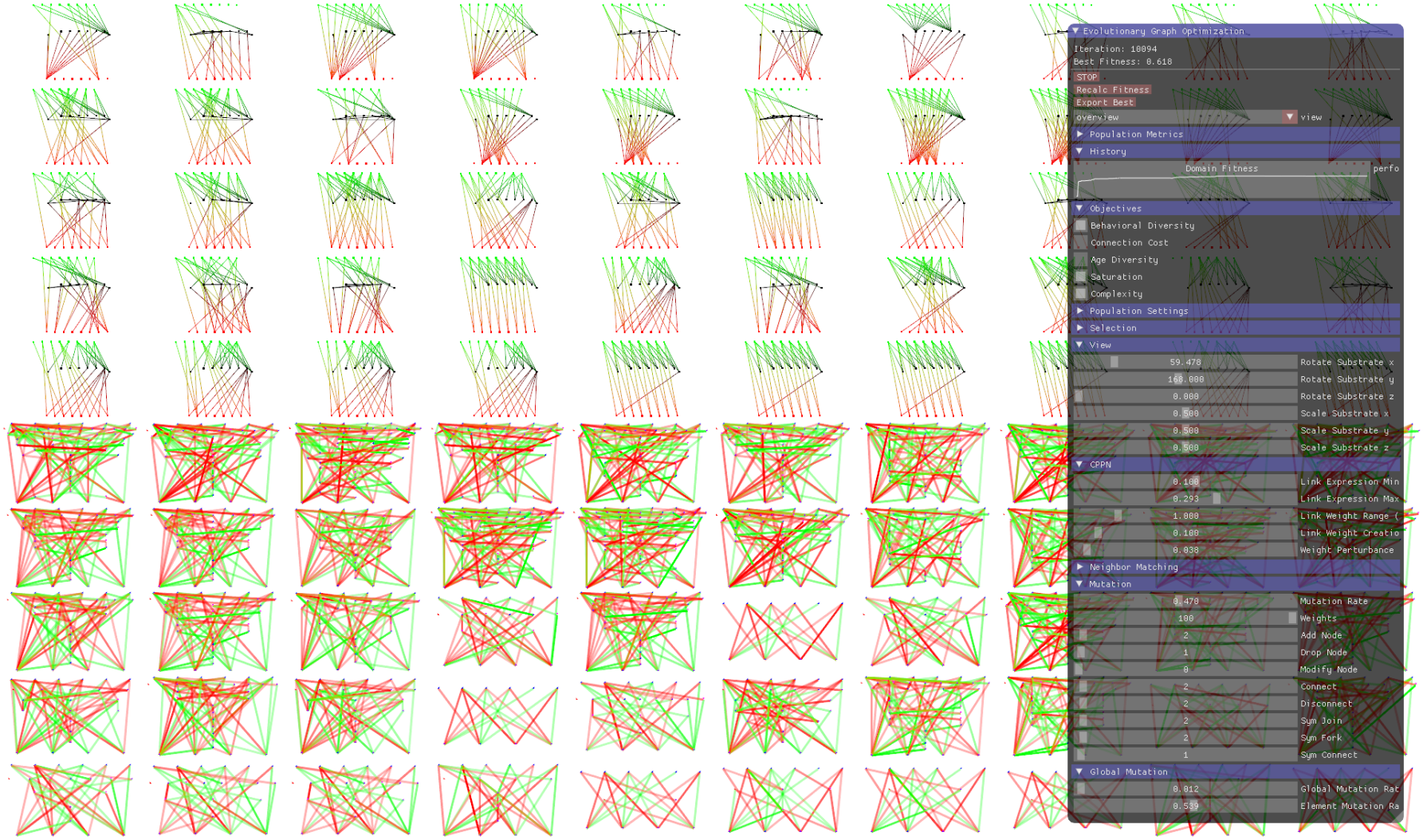


- N=100



- Complexity of CPPN independent of size of graph

Demo


Evolutionary Graph Optimization
Iteration: 10094
Best Fitness: 0.618
STOP
Recalc Fitness
Export Best
overview
view
Population Metrics
History
Domain Fitness
perfo
Objectives
Behavioral Diversity
Connection Cost
Age Diversity
Saturation
Complexity
Population Settings
Selection
View
59.478
Rotate Substrate x
168.000
Rotate Substrate y
0.000
Rotate Substrate z
0.500
Scale Substrate x
0.500
Scale Substrate y
0.500
Scale Substrate z
CPPN
0.100
Link Expression Min
0.293
Link Expression Max
1.000
Link Weight Range (
0.100
Link Weight Creatio
0.038
Weight Perturbance
Neighbor Matching
Mutation
0.470
Mutation Rate
100
Weights
2
Add Node
1
Drop Node
0
Modify Node
2
Connect
2
Disconnect
2
Sym Join
2
Sym Fork
1
Sym Connect
Global Mutation
0.012
Global Mutation Rat
0.539
Element Mutation Ra

Conclusion

Conclusion

- CPPN encode all aspects of graphs (edge/node weights, connectivity)

Conclusion

- CPPN encode all aspects of graphs (edge/node weights, connectivity)
- Patterns with following properties can be efficiently encoded:

Conclusion

- CPPN encode all aspects of graphs (edge/node weights, connectivity)
- Patterns with following properties can be efficiently encoded:
  - symmetry

Conclusion

- CPPN encode all aspects of graphs (edge/node weights, connectivity)
- Patterns with following properties can be efficiently encoded:
  - symmetry
  - imperfect symmetry

Conclusion

- CPPN encode all aspects of graphs (edge/node weights, connectivity)
- Patterns with following properties can be efficiently encoded:
  - symmetry
  - imperfect symmetry
  - repetition

Conclusion

- CPPN encode all aspects of graphs (edge/node weights, connectivity)
- Patterns with following properties can be efficiently encoded:
  - symmetry
  - imperfect symmetry
  - repetition
  - repetition with variation

Conclusion

- CPPN encode all aspects of graphs (edge/node weights, connectivity)
- Patterns with following properties can be efficiently encoded:
  - symmetry
  - imperfect symmetry
  - repetition
  - repetition with variation
- Functional dependencies of structure *and* weighting.

Conclusion

- CPPN encode all aspects of graphs (edge/node weights, connectivity)
- Patterns with following properties can be efficiently encoded:
  - symmetry
  - imperfect symmetry
  - repetition
  - repetition with variation
- Functional dependencies of structure *and* weighting.
- **For regular graphs the complexity of CPPNs in independent of the size of the graph.**

Conclusion

- CPPN encode all aspects of graphs (edge/node weights, connectivity)
- Patterns with following properties can be efficiently encoded:
  - symmetry
  - imperfect symmetry
  - repetition
  - repetition with variation
- Functional dependencies of structure *and* weighting.
- **For regular graphs the complexity of CPPNs in independent of the size of the graph.**
- Manual substrate placement and fixed number of nodes

Conclusion

- CPPN encode all aspects of graphs (edge/node weights, connectivity)
- Patterns with following properties can be efficiently encoded:
  - symmetry
  - imperfect symmetry
  - repetition
  - repetition with variation
- Functional dependencies of structure *and* weighting.
- **For regular graphs the complexity of CPPNs in independent of the size of the graph.**
- Manual substrate placement and fixed number of nodes
  - BUT: HyperNEAT-ES (Evolvable Substrate)

Karlsruhe Institute of Technology

Questions?

Lindenmayer-System

- Parallel string replacement system $G = \langle \Sigma, \omega, P \rangle$

Lindenmayer-System

- Parallel string replacement system $G = \langle \Sigma, \omega, P \rangle$
- Example:
  - $\Sigma = \{F, +, -\}$
  - $P = \{F \rightarrow F - F + +F - F\}$
  - $\omega = F + +F + +F$

Lindenmayer-System

- Parallel string replacement system $G = \langle \Sigma, \omega, P \rangle$
- Example:
  - $\Sigma = \{F, +, -\}$
  - $P = \{F \to F - F + +F - F\}$
  - $\omega = F + +F + +F$
  - $S_1 = F$

Lindenmayer-System

- Parallel string replacement system $G = \langle \Sigma, \omega, P \rangle$
- Example:
  - $\Sigma = \{F, +, -\}$
  - $P = \{F \rightarrow F - F + +F - F\}$
  - $\omega = F + +F + +F$
  - $S_1 = F$
  - $S_2 = F - F + +F - F$

Lindenmayer-System

- Parallel string replacement system $G = \langle \Sigma, \omega, P \rangle$
- Example:
  - $\Sigma = \{F, +, -\}$
  - $P = \{F \to F - F + + F - F\}$
  - $\omega = F + + F + + F$
  - $S_1 = F$
  - $S_2 = F - F + + F - F$
  - $S_3 = \overbrace{F - F + + F - F} - \overbrace{F - F + + F - F} + +$
    $\overbrace{F - F + + F - F} - \overbrace{F - F + + F - F}$

Lindenmayer-System

$$F \rightarrow F - F + +F - F$$

Koza (1991)

- Uses Genetic Programming

Koza (1991)

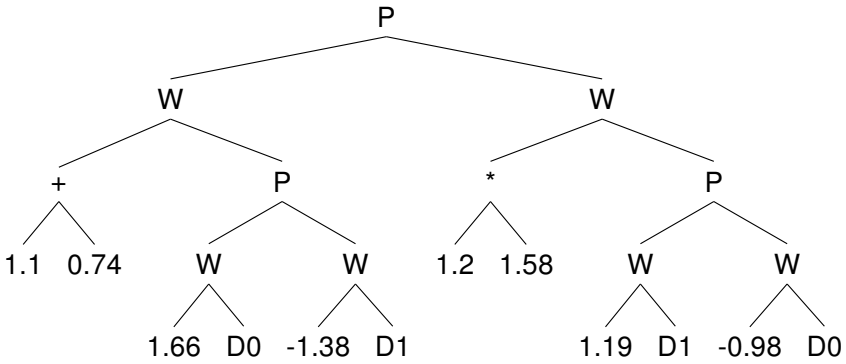
- Uses Genetic Programming
- Evolves LISP S-expression `(P (W (+ 1.1 0.74) ...))`

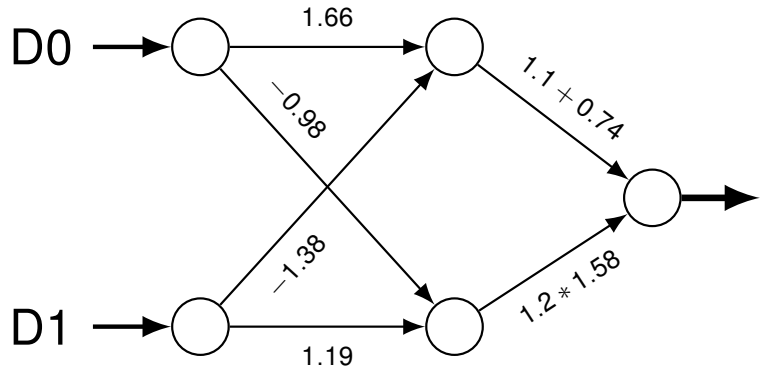
Koza (1991)

- Uses Genetic Programming
- Evolves LISP S-expression `(P (W (+ 1.1 0.74) ...))`
- Special symbols to represent network nodes

Koza (1991)

```
(P (W (+ 1.1 0.74) (P (W 1.66 D0) (W -1.38 D1)))
   (W (* 1.2 1.58) (P (W 1.19 D1) (W -0.98 D0))))
```





Koza (1991)

- Only acyclic networks with one output can be represented

Koza (1991)

- Only acyclic networks with one output can be represented
- Cannot encode repetitive patterns