On the Preservation of Input/Output Directed Graph Informativeness under Crossover

Andreas Duus Pape, J. David Schaffer, Hiroki Sayama, Christoper Zosh

November 9, 2025

Abstract

There exists a broad class of networks that connect inputs to outputs. These networks include chemical transformation networks, electrical circuits, municipal water systems, and neural networks. The goals of this paper are to provide a theoretical foundation for evolutionary crossover on this class of graphs and connect crossover to *informativeness*, a measure of the connectedness of inputs to outputs. Informativeness is defined as: a *partially informative* graph has at least one path from an input to some output, a *very informative* graph has a path from every input to some output, and a *fully informative* graph has a path from every input to every output. A neural network with non-zero weights and any number of layers is fully informative. As links are removed (assigned zero weight), it may become very, partially, or not informative. (The complement of informativeness is *actionability*, which is a measure of how connected outputs are from inputs.)

We define a crossover operation on IOD Graphs in which we find subgraphs with matching sets of forward and backward directed links to "swap." With this operation, IOD Graphs can be subject to evolutionary computation methods. We show that fully informative parents may yield a non-informative child. We also show that under certain conditions, crossover compatible, partially informative parents yield partially informative children, and very informative input parents with partially informative output parents yield very informative children. However, even under these conditions, full informativeness may not be retained. Similar results hold for actionability.

1 Introduction

There is a broad class of networks which connect inputs to outputs, used for modeling and problem solving in a variety of domains. Examples include chemical reaction networks (e.g., Unsleber and Reiher, 2020; Wen et al., 2023), municipal water systems (e.g., Shinstine et al., 2002; Wu and Clark, 2009), data flow networks (e.g., Meng et al., 2004), and electrical circuits (e.g., Koza et al., 1997;

Shanthi and Parthasarathi, 2009). We call these graphs Input/Output Directed Graphs or IOD Graphs. A common IOD Graph is the multi-layer perceptron, which is a feed-forward IOD Graph that converts inputs into "useful" outputs. Perceptrons are used to solve a variety of problems; for example, in cognitive science they are used to emulate human categorization behavior.[1]

While evolutionary operations like crossover have been used in all these domains, the application of crossover to IOD Graphs remains sparse. We provide a theoretical foundation for crossover across this class of networks. First, we establish how crossover can be applied to any two such networks. We also connect crossover to informativeness, a measure of the connectedness of inputs to outputs, and provide proofs putting bounds on how informativeness flows from parent networks to their children.

First, we formally define this class of graphs as Input/Output Directed Graphs or IOD Graphs: An *IOD Graph* is a graph with a set of nodes N and directed edges E, where N contains (a) a set of "input nodes" $I \subset N$, where each $i \in I$ has no incoming edges and any number of outgoing edges, and (b) a set of "output nodes" $O \subset N$, where each $o \in O$ has no outgoing edges and any number of incoming edges, and $I \cap O = \emptyset$. Nodes $n \in N$ which are neither inputs nor outputs, so $n \notin I, n \notin O$, are called "intermediate nodes."

Because of the prominence of neural networks, we will generally describe IOD Graphs as information flow networks and interpret the properties and results from that perspective. However, as mentioned above, these results about crossover could also apply to e.g. municipal water system design. Also, while many neural networks are feed-forward, meaning the networks do not contain any feedback loops, we do not restrict our study or the definition of IOD Graphs to feed-forward networks.

IOD Graphs solve problems by converting inputs into outputs, so an IOD Graph's ability to solve problems relies in part on the connectedness of inputs to outputs. We introduce a measure of the connected paths from inputs to outputs in 'informativeness.' *Informativeness* is a characterization of how information flowing into to the system is utilized, in the sense that it characterizes how many inputs are eventually connected to an output. More precisely, the *informativeness* of an IOD Graph is a characterization of how many paths exist from inputs to outputs. Informativeness is a key component of the value of an IOD Graph to solve a problem. At one extreme, an IOD Graph that has no path from an input to an output is useless at solving problems because it cannot deliver any information from the inputs to the outputs. On the other hand, an IOD Graph could have a path for every input and output pair, plus additional paths that interact, for example, as do the nodes in the layers of a multilayer perceptron.[2] (We also, as a corollary, introduce the concept of *actionability*,

[1] E.g., the classification learning literature beginning with (Shepard et al., 1961) and including the neural network ALCOVE (Kruschke, 1992). See more below.

[2] Informativeness is also applicable for these other types of IOD Graphs. For example, a municipal water system which has no paths from inputs to outputs would function very poorly

which is the symmetric measure of how many outputs are eventually connected from an input.)

One problem appropriate for IOD Graphs is the classification learning problem in cognitive science/psychology. A canonical version of this problem was by Shepard et al. (1961). In their laboratory, they showed human subjects objects characterized by a three-dimensional binary vector (e.g. big v. small, dark v. light, square v. triangle) and queried the category, A or B. The experimenter knew the true category, and told the human subject whether their guess was right; the error rate over time of different categorizations was measured (as the reader might imagine, some categorizations are easier to learn than others) and these experiments provide a data benchmark that future computational learning models strived to explain. One such explanation was ventured by Kruschke (1992), who approached this problem with a single-layer perceptron/feed-forward neural network with three inputs, two outputs, and nine nodes in the "hidden layer," for a total of 14 nodes. The set of all IOD Graphs with 14 nodes is vast: assuming at most one link between nodes, there are roughly $2^{132} \approx 5.4 \times 10^{39}$ different possible IOD Graphs with that number of inputs, outputs, and intermediate nodes. Growing potential solutions in this space via evolutionary methods seems wise. The DIVA model (Kurtz, 2007) is another IOD Graph in this literature, which would model the SHJ problem as 3 inputs and with two sets of 3 outputs for a total of 6 outputs. A crossover method like the one we describe in this paper could, for example, breed IOD Graphs from ALCOVE and DIVA implementations, possibly providing useful new solutions.

The purpose of this paper is to rigorously define a crossover operation and (begin to) characterize how informativeness and actionability change under this crossover operation. That is, we seek to lay a strong theoretical foundation for crossover on a broader class of graphs, of which feed-forward neural networks, municipal water systems, electric circuits, etc., are special cases, and relate these different methods to informativeness, which we believe helps capture some of the effectiveness of IOD Graphs to process information.

The crossover process described here is applied on the graph directly and does not use evo-devo. Evo-devo (or Evolutionary Development) is a gene-driven process in which a genome representation is translated into phenotypes, in this case, graphs, and crossover is done on the genomes, not the graphs themselves (Jacob, 1977; Gould, 1977). This direct crossover on network structure may foster Lamarckian evolution and may be compatible with some evo-devo crossover methods (see Section 2).

One application of this method is Spiking Neural Networks. Spiking Neural Networks (SNNs) are biologically-inspired neural networks that process information through discrete spikes or pulses over time (Hodgkin and Huxley, 1952; Maass, 1997). SNNs importantly take advantage of recurrent structures, and

as water could not flow.

we hypothesize that crossover on the network topology could help mitigate the competing conventions problem by preserving substructures of the network that are particularly useful in solving particular aspects of problems. Therefore, a topology-aware crossover method might preserve those structures and assist with genetic evolution in this space.

We describe relevant literature in Section 2. We formally define IOD Graphs and informativeness in Section 3. We formally define the crossover operation in Section 4, including the core definitions (Section 4.1) and a discussion of *crossover compatibility* (Section 4.2), which characterizes when two IOD Graphs can be subject to the crossover operation. We apply this crossover operation to informativeness in Section 5, 'The Preservation of Informativeness,' which proves the core theorems. Section 6 briefly extends the analysis from informativeness to a related idea of *actionability*, which characterizes the amount of informed behavior flowing out of a system, in the sense that it characterizes how many outputs are connected from an input. Section 7 discusses aspects of the results, such as the competing conventions problem and the distribution of informativeness categories across networks of different degree. Section 8 concludes.

2 Related Literature

The results in this paper concern a broad class of networks, Input/Output Directed Graphs, and a crossover operator that acts directly on these graphs without relying on an underlying genotype representation. To our knowledge, this is the first method which allows for crossover on the broad class of possibly recurrent IOD Graphs which operates on the topology directly. In what follows, we review existing approaches and highlight key ways in which our method differs.

First of all, our study is about a form of crossover. There is a large body of evolutionary computation work that does not use crossover, and only mutation; for example evolving the weights but not the topology of neural networks (e.g., Braun and Weisbrod, 1993; Braun and Ragg, 1997; Chandra et al., 2012). Methods without crossover are very different from our work here. The remaining literature consists of crossover methods with a genotype representation (Subsection 2.1), and crossover methods on non-recurrent or non-IOD Graphs (Subsection 2.2).

2.1 Crossover Methods with a Genotype Representation

Methods that apply crossover to genotype representations of networks, often called **Evo-devo** (short for "Evolutionary Development"), are gene-driven processes that work with genome representations (typically vectors of parameters) which can be converted into the corresponding phenotype the genes are meant to represent. In the context of this paper, we are interested in applications to evolving graphs (Jacob, 1977; Gould, 1977). In Evo-devo, all aspects of the evolutionary algorithm, including crossover, act on the genotype representations,

which are retained in the population. These genomes are then converted into their corresponding phenotypes (graphs) when the fitness of these genotypes are evaluated. All methods described in this section rely on a genotype representation, so they are distinct from our method in that way.

One strength of Evo-devo is that it is well defined for applications on IOD Graphs (e.g. Arifovic and Gencay, 2001; García-Pedrajas et al., 2006). One of the most famous genotype-based evolutionary network methods is NEAT (NeuroEvolution of Augmenting Topologies) and its extension HyperNEAT (Stanley and Miikkulainen, 2002; Stanley et al., 2009), which evolve network topologies and weights starting with minimal networks and growing them. There are genotype-based NEAT implementations which operate on recurrent networks (D'Ambrosio et al., 2014). Some studies, such as Dragoni et al. (2014) and Uriot and Izzo (2020), apply crossover to genomes in a network-aware method. In an approach most similar to ours, Uriot and Izzo (2020) build a representation between hidden layers of feed-forward neural networks based on functionality and applies crossover to that representation, while our crossover allows for arbitrary crossover cuts and for a larger class of graphs. One drawback of these methods, however, is that they can be susceptible to the problem of *competing conventions* (Schaffer et al., 1992). This occurs when fairly different genotype representations can correspond to similar or even the same phenotypes (graphs) and can lead offspring to inherit parts of incompatible solutions from their parents which cannot be easily reconciled.

In contrast with evo-devo, the crossover process described here is directly applied on the network topology and, therefore, is agnostic with respect to the genome representation. IOD Graph Crossover could lead to improved performance of evolution if the 'direct' representation of the graph fosters Lamarckian evolution (in which acquired traits can be inherited to offspring) through maintaining coherent sub-structures. This could be important for e.g. modifying trained neural networks.

Many existing evo-devo crossover processes will likely not achieve offspring consistent with IOD Graph Crossover as described in Definition 7. However, if a particular evo-devo process *could* be shown to obey all the properties in Definition 7, then such a process would follow all theorems here with regard to e.g. informativeness and actionability preservation, as well as any future theorems for IOD Graph crossover. By contrast, the crossover method in the study Uriot and Izzo (2020), described above, does not require preserving the number of links at the crossover as we require for crossover compatibility (Definition 4, Section 4). Therefore, the results in this paper do not apply.

2.2 Crossover Methods on non-recurrent or non-IOD Graphs

We discuss the literature on the application of crossover directly on the networks (without a genotype representation), that are on either graphs which are not IOD Graphs or they are a non-recurrent subset of IOD Graphs.

First, we note that there are a number of applications where non-IOD Graph networks are grown or modified. Non-IOD Graphs of interest include social networks and networks of webpages. In a social network, each node is both an input and an output, which is explicitly ruled out in IOD Graphs (where inputs and outputs are disjoint sets). In a network of webpages, where links are hyperlinks, no webpage is an input nor an output. Therefore, none of the properties attributed to IOD Graphs, such as those established in this study, necessarily apply to social networks or networks of linked webpages.

Social networks are often sampled from data instead of grown as we do here (e.g., Lotf et al., 2022), although there are studies in which virtual social networks are grown using evolutionary methods (e.g., Topirceanu et al., 2015).

One literature on networks of webpages discusses Google Pagerank (e.g., Bianchini et al., 2005; Langville and Meyer, 2006; Ipsen and Selee, 2008; Sallinen et al., 2023). This literature uses a term "dangling nodes" to refer to webpages with incoming, but no outgoing hyperlinks. In this paper, we use "dangling node" to refer to nodes which do not lie on a path between inputs and outputs (see Section 4). Our usages are very similar: in both cases they mean nodes which lead nowhere. Note that inputs and outputs are not defined on networks of webpages, but if one were to define some webpages as inputs and some as outputs, a dangling node as they define it would also be a dangling node as we define it here.

Evolutionary Programming evolves function trees using evolutionary algorithms like the genetic algorithm to solve problems (Koza, 1999). Function trees are a type of IOD Graph which has a feed-forward, tree-like structure. This structure is natural for applying crossover: any "branch" can be trimmed and swapped with another "branch." Trees are, importantly, non-recurrent.

Since all neural networks are IOD Graphs, this paper is a contribution to the substantial research effort applying crossover to neural networks. Starting with the perceptron (Rosenblatt, 1957), the majority of neural networks explored, including modern deep nets, are feed-forward networks. While more sophisticated computation is possible when networks include recurrent links, there has yet to emerge a comprehensive theory of how the computational power of networks with recurrent links can be achieved. There has been much empirical research aimed at evolutionary computation for this task; for a recent summary, see Stanley et al. (2019). **Neural Architecture Search (NAS)** and **Evolutionary Deep Learning (EDL)** are a set of methods for designing neural networks for particular goals automatically/algorithmically, for example in the domain of image classification. See Elsken et al. (2019) for a survey of NAS. These models are related to our work, in that they preform evolutionary operations on networks. However, they are distinguished in two major ways: first of all, this literature focuses on non-recurrent networks, i.e., for example, Deep Neural Networks. Second, evolutionary operations generally use mutation and not crossover, with some exceptions. Sun et al. (2019a) use a random forest method, which slices the network into sub-trees for swapping, which is not well-defined on recurrent

networks. Sun et al. (2019b) is built on network 'blocks,' which are a series of input-to-output subprocesses in the overall network. Here, crossover appears the most similar to ours, in which all inputs are separated from all outputs on the crossover surfaces. However, these are again only defined on non-recurrent networks. Moreover, crossover is limited to swapping blocks, with no notion of cutting between the blocks. Because of this, they do not have nor do they need a corresponding notion of crossover compatibility, electing instead to modify the networks so they connect. He et al. (2021) uses a cell-based architecture in which subsets of the graph nodes are designated cells (Zoph et al., 2018), that these authors allow to be "swapped" in crossover. These also appear to be defined only on non-recurrent networks and do not have crossover which separates inputs from outputs.

3 Definitions

Definition 1 *An Input/Output Directed Graph or* IOD Graph *is a graph with a set of nodes N and directed edges E, such that N contains two subsets:*

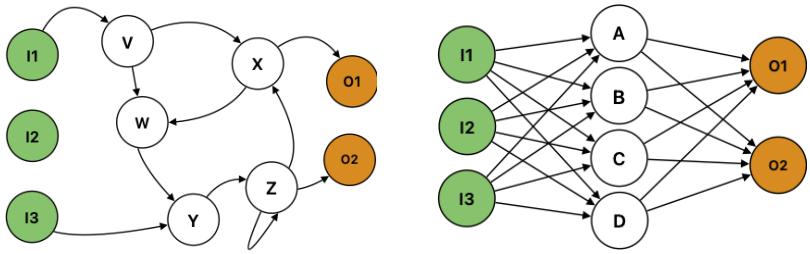
- *A set of nodes $I \subset N$, $|I| > 0$, where each $i \in I$ has no incoming edges and any number of outgoing edges, and*
- *A set of nodes $O \subset N$, $|O| > 0$, where each $o \in O$ has no outgoing edges and any number of incoming edges, and where $O \cap I = \emptyset$*

The set I are inputs into the system and the set O are outputs. We name the nodes which are neither inputs nor outputs as intermediate nodes. Specifically, $N \setminus (I \cup O)$ is the set of intermediate nodes. We do not require the set of intermediate nodes to be non-empty.

Figure 1a depicts an example IOD Graph. A perceptron with any number of layers is an example of an IOD Graph; Figure 1b depicts a single-layer perceptron. Perceptrons are used to solve various problems relating input variables to output variables. We consider IOD Graphs as a larger class of networks that could plausibly be used to solve those same kinds of problems.

One property we wish to define on IOD Graphs is how the inputs are connected to the outputs via directed paths. We define *informativeness*, which involves the connections via directed paths from the input nodes to the output nodes: A *partially informative IOD Graph* has at least one path from an input to an output, a *very informative IOD Graph* has a path from every input to an output, and a *fully informative IOD Graph* has a path from every input to every output. On the other extreme, a *non-informative IOD Graph* has no paths from inputs to outputs. A non-informative IOD Graph is worthless as a problem-solving engine.

Figure 1a is a *partially informative graph*: For example, $I1 \to V \to W \to Y \to Z \to O2$ is a directed path which leads from $I1 \in I$ to $O2 \in O$. However, it is not *very informative*, because there is no path from $I2$ to any output, and



(a) A Partially Informative IOD Graph (b) A Fully Informative IOD Graph (A Single-layer Perceptron)

Figure 1: Two Input/Output Directed Graphs of Varying Informativeness. *Partially informative IOD Graphs have at least one path from an input to an output. Fully informative IOD Graphs have a path from every input to every output.*

it is not fully informative because very informative is a necessary condition for *fully informative*. If the IOD Graph were modified by adding a directed edge from $I2 \rightarrow W$, then this modified IOD Graph would be very informative. A multi-level perceptron with non-zero weights on its edges, for example depicted in Figure 1b, is fully informative. Every input has a directed path to every output.

4 The Crossover Operation

4.1 Crossover-related Definitions

We define a crossover operation on IOD Graphs in which we find subgraphs with matching sets of forward and backward directed links to "swap." Here we define the relevant terms to fully define the crossover operation. In the genetic algorithm, the one point crossover operation splits the two parent genomes, and then splices the initial part of one parent's genome and the latter part of the other parent's genome. Similarly, we split two IOD Graphs into an input part and an output part and then splice the input part of one IOD Graph to the output part of the other.

In order to define this operation, we must define *first*, how to split an IOD Graph, and *second,* how to splice them.

The natural way to split an IOD Graph is to partition its nodes into two non-overlapping sets, where all input nodes are in one part and all output nodes are in the other part.[3] We define an *IO Partition* as follows:

Definition 2 *Let G be an Input/Output Directed Graph, with corresponding subsets of nodes I and O. Then define an* IO Partition *of G to be a partition*

[3]Definition: A *partition* (S_1, S_2) of the set S requires that $S_1 \cup S_2 = S$ and $S_1 \cap S_2 = \emptyset$.

(Ψ, Ω) *of* N*, where* $I \subseteq \Psi$ *and* $O \subseteq \Omega$*.*

For any IO Partition (Ψ, Ω), we refer to Ψ as the input part and Ω the output part of the partition.

Given this definition of an IO partition, we define $M(\Psi, \Omega)$ as the set of all edges which connect the parts of the partition. This is called a *cut* in network theory. Using (n_1, n_2) to denote a directed edge from n_1 to n_2 and E to denote the list of all edges, we can formally define $M(\Psi, \Omega)$ as follows:

For all $e \in E$, where $e = (n_1, n_2)$ or $e = (n_2, n_1)$

$$M(\Psi, \Omega) = \{e \quad | \quad n_1 \in \Psi, n_2 \in \Omega\}$$

We call $M(\Psi, \Omega)$ the *separating membrane* associated with IO Partition (Ψ, Ω). An IO Partition and corresponding separating membrane are depicted in Figure 2.

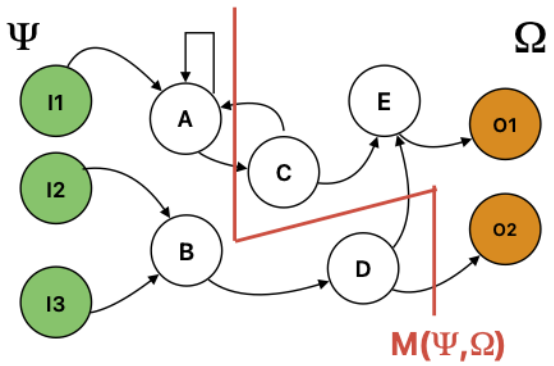


Figure 2: An IO Partition and corresponding membrane.
An IO Partition separates inputs from outputs. The membrane is the set of links which cross the partition sets.

Each $e \in M(\Psi, \Omega)$ either connects from the input part to the output part or the reverse. If it connects from the input part to the output part, we call it a *forward link*, because it links from inputs to outputs. If instead it connects from the output to the input, we call it a *backwards link*. Let $F(\Psi, \Omega)$ be the set of forward links in a membrane $M(\Psi, \Omega)$ and $B(\Psi, \Omega)$ be the set of backwards links in $M(\Psi, \Omega)$. Clearly, $M(\Psi, \Omega) = F(\Psi, \Omega) \cup B(\Psi, \Omega)$.

We also wish to have a way to describe a partition as cohering within the input or output parts.

Definition 3 *For an IO Partition* (Ψ, Ω)*:*

- The input part Ψ is contiguous *(or, equivalently,* (Ψ, Ω) is input-contiguous*) if, for every* $n \in \Psi \backslash I$*, there is a path from some input* $i \in I$ *to* n *which lies entirely within* Ψ*, and*

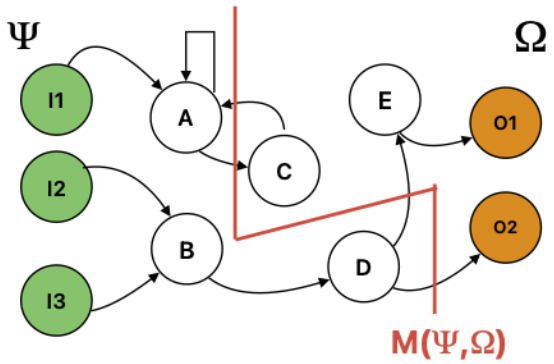


Figure 3: This IO Partition is input-contiguous but not output-contiguous
All nodes in the input part connect from an input without leaving the part. At least one node on the output part is not connected to an output in this way.

- The output part Ω is contiguous *(or, equivalently,* (Ψ, Ω) is output-contiguous*) if, for every* $n' \in \Omega \backslash O$*, there is a path from* n' *to some output* $o \in O$ *which lies entirely within* Ω*.*

If both the input part Ψ *and output part* Ω *are contiguous, we say* the IO partition (Ψ, Ω) is contiguous.

Requiring a partition to be contiguous means intermediate nodes in the input part are connected from inputs and intermediate nodes in the output part are connected to outputs. Figure 3 depicts an IOD Graph and IO Partition which is input-contiguous but not output-contiguous. Output contiguousness fails because there is no path from C to an output which is contained in Ω.

4.2 Crossover compatibility

Suppose G, G' are IOD Graphs with IO Partitions (Ψ, Ω) and (Ψ', Ω'). Then we say these IO Partitions are *crossover compatible* if they have the same number of forward and backward links, and the rather technical assumption (typically easily satisfied) that the input part of one parent shares no nodes with the output part of the other and vice versa.

Definition 4 *IOD Graphs and IO Partitions* $\{G, (\Psi, \Omega)\}$ *and* $\{G', (\Psi', \Omega')\}$ *are* crossover compatible *if* $\Psi \cap \Omega' = \emptyset$*,* $\Psi' \cap \Omega = \emptyset$*, and*

$$|F(\Psi, \Omega)| = |F(\Psi', \Omega')|, \text{ and}$$
$$|B(\Psi, \Omega)| = |B(\Psi', \Omega')|$$

If IO Partitions (Ψ, Ω) and (Ψ', Ω') are crossover compatible, then they can be used to create crossover children. For simplicity, and without lack of generality, we define the crossover child constructed from crossover compatible Ψ and Ω'.

In this case, we call G the input parent, because it contributes the input part, and G' the output parent, because it contributes the output part.[4] First, we define a crossover membrane $\hat{M}$:

Definition 5 *Suppose G, G' are IOD Graphs with crossover compatible IO Partitions (Ψ, Ω) and (Ψ', Ω'). Then a (G, G') crossover membrane $\hat{M}$ is defined as as all edges f'' and b'', where*

1. *For each forward edge $f \in F(\Psi, \Omega)$, select, without replacement, a forward edge f' in $F(\Psi', \Omega')$, and then construct the edge f'' which connects the source of f to the destination of f', and*
2. *For each backward edge $b \in B(\Psi, \Omega)$, select, without replacement, a backward edge b' in $B(\Psi', \Omega')$, and construct the edge b'' which connects the source of b' with the destination of b.*

Note that a given pair of crossover compatible IO Partitions (Ψ, Ω) and (Ψ', Ω'), there must exist at least one crossover membrane, as shown in Lemma 1. There may be, and often are, more than one possible crossover membrane. In that case, Ψ and Ω' can be connected in meaningfully different configurations.

Lemma 1 *Suppose G, G' are two crossover compatible IOD Graphs. Then there exists a (G, G') crossover membrane.*

Proof. Since G, G' are crossover compatible, then select crossover compatible IO Partitions (Ψ, Ω) for G and (Ψ', Ω') for G'.

Then define f, f' and b, b' as the sets of forward and backward links; namely let f, f' be $f = F(\Psi, \Omega)$ and $f' = F(\Psi', \Omega')$; and let b, b' be $b = B(\Psi, \Omega)$ and $b' = B(\Psi', \Omega')$.

By definition of crossover compatible, $|f| = |f'|$ and $|b| = |b'|$. Create two one-to-one and onto mappings $g : f \to f'$ and $h : b' \to b$.

For a link l, define l_{source} as the source node of l and let l_{dest} as the destination node of l; so by definition each link $l = (l_{\text{source}}, l_{\text{dest}})$.

Then define $\hat{M}_f$, $\hat{M}_b$ as:

$$\hat{M}_f = \{(x_{\text{source}}, g(x)_{\text{dest}}) \quad | \quad x \in f\}$$
$$\hat{M}_b = \{(y_{\text{source}}, h(y)_{\text{dest}}) \quad | \quad y \in b'\}$$

$\hat{M}_f \cup \hat{M}_b$ is a (G, G') crossover membrane. ■

Now we define the child of crossover:

[4] Notably, there may also be a crossover child that can be constructed from (Ψ', Ω). The process and discussion is identical with the roles of Ψ and Ψ', and Ω' and Ω, reversed.

Definition 6 *Suppose G, G' are IOD Graphs with crossover compatible IO Partitions (Ψ, Ω) and (Ψ', Ω'). Then construct a crossover child C as the graph consisting of the following nodes and edges:*

- *The set of nodes in C is $\Psi \cup \Omega'$*
- *The set of edges in C is the union of:*
  - *the set of edges e which connect elements of Ψ,*
  - *the set of edges e' which connect elements of Ω', and*
  - *a crossover membrane $\hat{M}$*

With this definition in hand, we can define *IOD Graph crossover*:

Definition 7 *IOD Graph Crossover is any algorithm which transforms any two IOD Graphs with crossover compatible IO Partitions into a crossover child.*

Crossover is depicted in Figure 4. On the left, we have the Input Parent IOD Graph. The set of nodes to the left of $M1$ are the set Ψ, i.e. $\Psi = \{I1, I2, I3, A, B, C\}$. The set of nodes to the right are the set $\Omega = \{C, E, O1, O2\}$. The membrane $M1 = M(\Psi, \Omega)$ contains three forward links and one backward link: $F(\Psi, \Omega) = \{(A, C), (D, E), (D, O2)\}$, $B(\Psi, \Omega) = \{(C, A)\}$

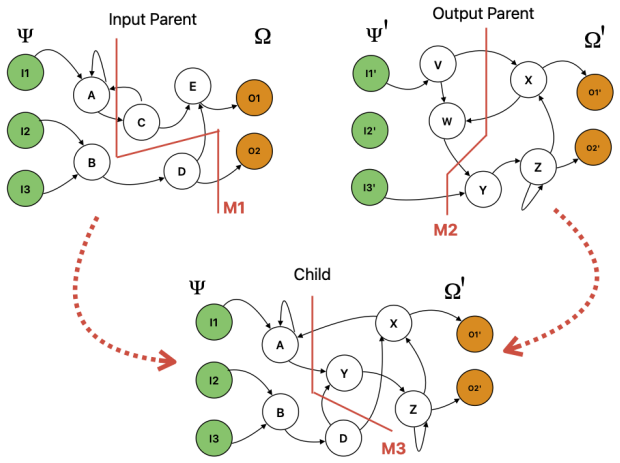


Figure 4: The Crossover Operation on Input/Output Directed Graphs
This figure depicts a child network constructed from an input part and an output part of two different, crossover compatible parent networks.

On the right, we have the Output Parent IOD Graph. $\Psi' = \{I'1, I'2, I'3, V, W\}$ and $\Omega' = \{X, Y, Z, O1, O2\}$. Like $M(\Psi, \Omega)$, $M(\Psi', \Omega')$ contains three forward links and one backward link: $F(\Psi', \Omega') = \{(V, X), (W, Y), (I3, Y)\}$, $B(\Psi', \Omega') = \{(X, W)\}$. The Input Parent and Output Parent are crossover compatible because the membranes $M1 = M(\Psi, \Omega)$ and $M2 = M(\Psi', \Omega')$ have the same

number of forward links (three) and the same number of backward links (one). Since they are crossover compatible, we can construct a crossover membrane $M3$ which connects the input part of the Input Parent (set Ψ) to the output part of the Output Parent (set Ω'). In Figure 4, we depict crossover membrane $M3 = \{(A, Y), (D, Y), (D, X), (X, A)\}$. As with $M(\Psi, \Omega)$ and $M(\Psi', \Omega')$, $M3$ contains three forward and one backward link. Given this crossover membrane, the Child is a well-defined IOD Graph.

As in this, crossover child C of two IOD Graphs is always an IOD Graph. That is, the set of IOD Graphs is closed under crossover. The proof of this claim is straightforward. We state this formally in Lemma 2:

Lemma 2 *If G and G' are IOD Graphs, then any child produced by crossover will also be an IOD Graph.*

Proof. Definition 6 implies that the crossover child C is a graph (that is, a set of nodes and a set of edges). Moveover, it contains a set of inputs $I \subseteq \Psi$, where all nodes $i \in I$ have no incoming links, and a set of outputs $O \subseteq \Omega$, where are notes $o \in O$ have no outgoing links. Since $\Psi \cap \Omega = \emptyset$, then $I \cap O = \emptyset$. Therefore the child C is an IOD Graph. ■

The reader may be interested in applications to feed-forward networks such as neural networks or perceptrons. Like the set of IOD Graphs, the set of feed-forward networks is closed under crossover, as shown in Lemma 3:

Lemma 3 *If G and G' are feed-forward IOD Graphs, then any child produced by crossover will also be a feed-forward IOD Graph.*

Proof. Suppose two feed-forward IOD Graphs G and G' produce a child which is not feed-forward. Then there is an edge which causes a loop in a child which was not present in either parent. This loop cannot involve only nodes within Ψ nor nodes only within Ω', because G and G' are feed-forward networks. This loop must have been "created" by the crossover. This means it must contain one forward and one backward link from the crossover membrane. However, the crossover membrane contains no backward links. Contradiction. ■

While the set of feed-forward networks is closed under crossover, the set of multilayer perceptrons is not; crossover can produce children who do not maintain the multilayer structure. Crossover between perceptrons *can* be guaranteed to produce perceptrons if the set of IO Partitions is restricted such that all nodes in the same layer are in the same part of the partition (e.g. if a node $n \in \Psi$ then all $\tilde{n}$ in the same layer as n are also in Ψ). This proof is omitted.

Note that the parents in Figure 4 share the same number of inputs and the same number of outputs. This is a standard case for most applications, in which there is a fixed set of inputs and outputs that are "shared" in some sense by all candidate networks. Though that is the typical case, it is not necessary for the mathematical results presented here, so we do not require it. Figure 7, for example, depicts a crossover in which parents do not match in inputs or outputs. That kind of crossover may be relevant for a decision maker in

an evolving setting; for example, a robot who loses a sensor or receives a new actuator as an upgrade.

5 The Preservation of Informativeness

There is no guarantee that informativeness of two IOD Graphs are preserved under crossover.

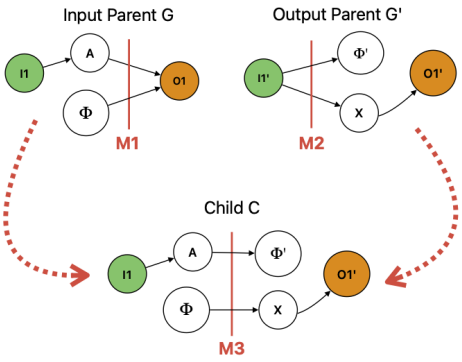


Figure 5: Two fully informative parents may yield a non-informative child. *Each parent has a path leading from the input to the output; however the child does not. This is a simple depiction of this issue.*

Figure 5 illustrates an example with two fully informative parents that each have one input and one output, and a path which connects them. In addition, input parent G has what we call a 'false input' ϕ and the output parent has a corresponding 'false output,' ϕ'. The figure illustrates how a false input/false output pair can completely disable a path from an input to an output in the child. The crossover operation acts like flipping a switch, breaking flow of information.[5,6]

The false output ends the flow of information from the input. This 'causes' non-informativeness. However, a false output alone is not sufficient for this example. The false input is also required. Without a false input in Input Parent G, these IO Partitions would not be crossover compatible and therefore crossover could not be performed. In particular, $|F(\Psi, \Omega)|$ would be two, while $|F(\Psi', \Omega')|$ would be one. Node X would require an incoming link from the membrane but there would be no link to be found.

The process shown in Figure 5 generalizes exponentially with inputs and outputs, as shown in Theorem 1:

[5]Note: there is an alternative crossover membrane which would preserve full informativeness: namely the links (A, X) and (ϕ, ϕ') comprise a crossover membrane which results in a fully informative child.

[6]Later, we say that ϕ and ϕ' are *dangling nodes*. See the *no dangling nodes condition* below (Definition 8).

Theorem 1 *The child of two IOD Graphs may retain no informativeness of the parents.*

Proof. We show a child of two IOD Graphs may retain no informativeness of the parents by creating two IOD Graphs of arbitrary informativeness, then showing they can produce a crossover child which has no informativeness.

Suppose G and G' are IOD Graphs which each have J inputs I, I' and K outputs O, O'. G is the input parent and G' the output parent.

Suppose G has the following structure:

For each input node $i \in I$, there are K paths, called $p(i,k)$, which may have intermediate nodes in common. Path $p(i,k)$ leads from input i to either output k or an intermediate node with no outgoing links. If every path $p(i,k)$ leads from input i to output k, $\forall i,k$, then the IOD Graph is fully informative. The more paths end in intermediate nodes, the lower the informativeness. Therefore, any level of informativeness can be achieved by the choice of paths $p(i,k)$ in the construction of G.

In addition to these paths, G has $J \cdot K$ nodes which we will call "false inputs." We name these nodes $\phi(i,k)$, for $i = 1,\ldots,J, k = 1 \ldots K$. False input $\phi(i,k)$ has no incoming edges and one outgoing edge, which connects directly to output k. $\phi(i,k)$ is the false input we will use to replace input i.

G has no other nodes or edges other than those described above.

The IO Partion (Ψ, Ω) associated with G is:

$$(\Psi = N \backslash O, \Omega = O)$$

Suppose G' has the following structure:

For each input node $i' \in I'$, there are K paths, called $p'(i', o')$. Path $p'(i', o')$ leads from input i' to either output o' or an intermediate node with no outgoing links. As above, these paths may have nodes in common. Like in graph G on the input part, IOD Graph G' can achieve any level of informativeness.

G' has $J \cdot K$ nodes which we will call "false outputs." We name these nodes $\phi'(i', o')$, for $i' \in I', o' \in O'$. False output $\phi'(i', o')$ has one incoming edge and no outgoing edges. It is connected directly from input i'. $\phi'(i', o')$ is the false output we will use to replace output o'.

Like IOD Graph G, G' has no other nodes or edges other than those described above.

The IO Partion (Ψ', Ω') associated with G' is:

$$(\Psi' = I, \Omega' = N \backslash I)$$

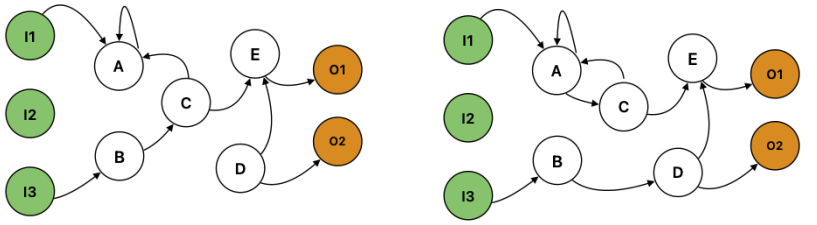
We then construct a crossover membrane $\hat{M}$ which breaks all informative paths by connecting potential informative paths to false inputs and outputs:

First we create a one-to-one and onto mapping from I to I', called $j : I \to I'$, and a one-to-one and onto mapping O to O' called $k : O \to O'$.

Second, for each $i \in I$ and $o \in O$, $\hat{M}$ does the following:

- it connects path $p(i, o)$ to false output $\phi'(j(i), k(o))$, and
- it connects each false input $\phi(i, k)$ to path $p'(j(i), k(o))$.

Then in the resulting child, every path leading out of each input $i \in I$ leads to false output $\phi'(\cdot)$. This means that the IOD Graph has no informativeness. ■



(a) Nodes A and D Dangle (b) No Dangling Intermediate Nodes

Figure 6: The No Dangling Nodes Condition.
Dangling nodes are those not on a path from an input to an output. On the left, A has no outgoing link. D has no incoming link.

False inputs and false outputs are what we call 'dangling nodes.' Dangling nodes are intermediate nodes which do not lie on a path from an input to an output.[7] To preserve informativeness through inheritance, we use the property which rules out dangling nodes, the *no dangling nodes condition*:

Definition 8 *An IOD Graph satisfies the* no dangling nodes condition *if every intermediate node is on a path from an input $i \in I$ to an output $o \in O$.*

In Figure 6a, both nodes A and D are "dangling" because neither is on a path from an input to an output. Figure 6b satisfies the No Dangling Node condition because every intermediate node falls on a path from an input to an output. Node $I2$ is not on a path to an output, but it is an input, not an intermediate node, so does not violate the condition.

It follows immediately that IOD Graphs which satisfy the No Dangling Nodes condition and have a non-empty set of intermediate nodes are partially informative. This is because any existing intermediate node must lie on a path from an input to an output, and therefore such a path exists. However, an IOD Graph can satisfy the no dangling nodes condition and not be very informative, because there can be inputs which connect to nothing, as seen in Figure 6b.

[7] For example, in Figure 5, which illustrates full informativeness not being preserved, nodes ϕ and ϕ' dangle.

Preservation of the No Dangling Nodes condition under inheritance is the cornerstone of informativeness preservation. It turns out that *contiguousness* of parents' IO Partitions is enough to guarantee that the no dangling condition is inherited by the child. (In particular, the input part must be contiguous and the output part must be contiguous.) Figure 7 illustrates an example of why this property delivers inheritance of the No Dangling Nodes condition. Below, Theorem 2 states the claim formally and proves it.

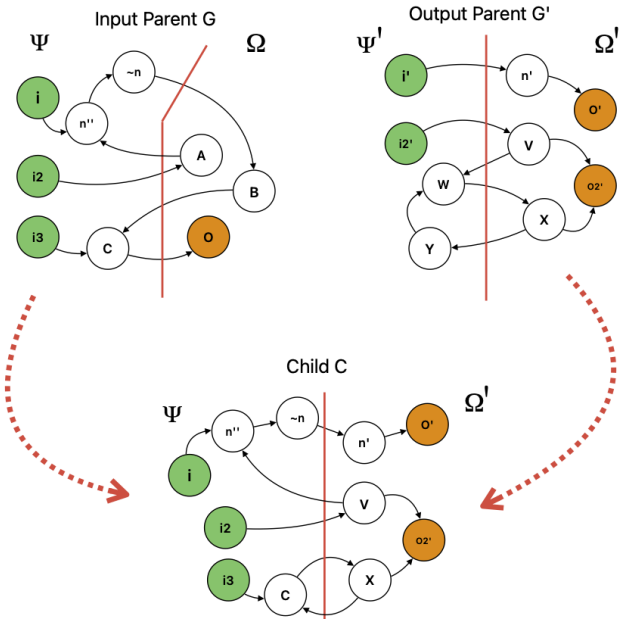


Figure 7: No Dangling Nodes Condition Inheritance: An Example
This is an illustration of why crossover preserves the no dangling node condition. See text.

In Figure 7, suppose that the node n'' is an arbitrary node in the input part Ψ. We seek to show that n'' must be on a path from an input to an output in the child C. In this example, that path turns out to be $i \to n'' \to\sim n \to n' \to o'$. We build that path in pieces: pieces p, q, and r, which we define below.[8]

Consider the input parent G. Since G is input-contiguous, we are assured of the path from input i to node n'', which we call path p. Path p lies entirely within Ψ. Since G satisfies the No Dangling Nodes condition, we are also assured of a path from $i2$ to O. In this case, the path is $i2 \to A \to n'' \to\sim n \to B \to C \to O2$. This path was selected to illustrate that the path may cross between Ψ and Ω

[8]Note that the parents in Figure 7 do not share the same number of inputs nor the same number of outputs. This is not necessary for the mathematical results presented here, so we do not require it. Please see Section 4.2 for details.

multiple times; in particular, we are not assured from No Dangling Nodes alone that the subpath from $i2$ through $\sim n$ to n'' resides entirely within Ψ.

There is a part of the path $i2 \rightarrow A \rightarrow n'' \rightarrow\sim n \rightarrow B \rightarrow C \rightarrow O2$ which we use later: the subpath $n'' \rightarrow\sim n$, which we call path q.

Now consider the output parent G' in Figure 7. Here, the No Dangling Nodes condition assures us that this $n' \in \Omega'$ that has been selected must have some path to some output o' that lies entirely within Ω', which we call path r.[9]

Note that the IO Partitions of G and G' depicted in Figure 7 are crossover compatible because $|F(\Psi, \Omega)| = |F(\Psi', \Omega')| = 3$ and $|B(\Psi, \Omega)| = |B(\Psi', \Omega')| = 2$.

Consider now Child C, we see the path $p \rightarrow q \rightarrow r$ which is $i \rightarrow n'' \rightarrow\ n \rightarrow n' \rightarrow o'$. This is a path which satisfies that node n'' lies on a path from an input to an output in Child C. This demonstrates that No Dangling Nodes is inherited.

Theorem 2 *Suppose IOD Graph G has input-contiguous IO partition (Ψ, Ω) and IOD Graph G' has output-contiguous IO partition (Ψ', Ω'), and further suppose both graphs G and G' satisfy the no dangling nodes condition. Then any crossover child produced by $(G, (\Psi, \Omega))$ and $(G', (\Psi', \Omega'))$ must also satisfy the no dangling nodes condition.*

Proof. Suppose $(G, (\Psi, \Omega))$ and $(G', (\Psi', \Omega'))$ are crossover compatible. Let the nodes of G include input nodes I and output nodes O and the nodes of G' include of input nodes I' and output nodes O'.

Let $\hat{M}$ be a crossover membrane that is used to construct a crossover child C. Now, all nodes of C must belong to Ψ or Ω'.

Suppose there are no intermediate nodes in C. Then the result is immediate.

Now suppose there is at least one intermediate node in C. Consider an arbitrary intermediate node in C.

Case 1. Suppose that intermediate node $n'' \in \Psi \backslash I$.

Since the IO Partition (Ψ, Ω) is contiguous, there must be a path from some $i \in I$ to n'' which is contained within Ψ. We call this path $p = (i, \ldots, n'')$.

Now, because G satisfies the no dangling nodes condition, n'' in G must be on a path from $(\tilde{i}, \ldots, n'', \ldots, \tilde{o})$ for some $\tilde{i} \in I, \tilde{o} \in O$.[10] Consider the subpath $(n'', \ldots, \tilde{o})$. Now that path must contain a node in Ω. Find the first node in the path $(n'', \ldots, \tilde{o})$ which is in Ω, and consider the node $\tilde{n}$ which links to that node. That is the path $(i, \ldots, n'', \ldots, \tilde{n})$. We define the subpath $q = (n'', \ldots, \tilde{n})$. Note

[9] Strictly speaking, No Dangling Nodes is not required for the Output Parent G' in this part of the proof. It is required later, when we consider $n'' \in \Omega'$.

[10] Note that the initial segment of this path $(\tilde{i}, \ldots, n'')$ need not lie within Ψ, which is why the earlier path segment p is needed.

that q resides entirely within Ψ. This means the path $p \to q$ resides entirely within Ψ.

In C, $\tilde{n}$ must link to some $n' \in \Omega'$ by construction of the crossover membrane $\hat{M}$. Since the IO partition (Ψ', Ω') is contiguous, there must be a path from n' to an output $o' \in O'$ fully contained in Ω, called path $r = (n', \ldots, o')$. Since p, q, and r reside in C, then the overall path

$$p \to q \to r = (i, \ldots, n'', \ldots, \tilde{n}, n', \ldots, o')$$

must be in C and therefore the no dangling node condition is satisfied for n''.

Note: It is possible in the previous paragraph that n' may be an output, i.e. $n' \in O'$. In this case the corresponding path R is the trivial $r = (n')$ and the rest of the proof follows: path $(i, \ldots, \tilde{n}, n')$ is a path which is in C and, since $n' \in O'$, it is a path which connects inputs to outputs.

Case 2. Suppose that the intermediate node $n'' \in \Omega \backslash O'$

A symmetric argument applies, stated here for completeness: Since the IO Partition (Ψ', Ω') is contiguous, there must be a path from n'' to some $o' \in O'$ which is contained within Ω'. We shall call this path $p' = (n'', \ldots, o')$. Now, n'' in G' is on a path from $(\bar{i}, \ldots, n'', \ldots, \bar{o})$ for some $\bar{i} \in I', \bar{o} \in O'$. Consider the path $(\bar{i}, \ldots, n'')$. Now that path must contain a node in Ψ'. Find the last node in the path $(\bar{i}, \ldots, n'')$ which is in Ψ, and consider the node $\bar{n}$ which links from that node. This is the path $(\bar{n}, \ldots, n'', \ldots, \bar{o})$. We name the subpath $q' = (\bar{n}, \ldots, n'')$. Note that q' resides entirely within Ω' and therefore $q' \to p'$ also resides entirely within Ω. In C, there must be some some $n \in \Psi$ which links to $\bar{n}$. By the IO partition (Ψ, Ω) being contiguous, there must be a path an input $i \in I$ to n fully contained in Ψ, which we call $r' = (i, \ldots, n)$. Therefore the path

$$r' \to q' \to p' = (i, \ldots, n, \bar{n}, \ldots, n'', \ldots, \bar{o})$$

must be in C and therefore the no dangling node condition is satisfied for n''.

Given Case 1 and Case 2, we have demonstrated that all intermediate nodes in C must lie on a path in C from some input in I to some output in O'. Therefore, the no dangling nodes condition is satisfied. ■

When Theorem 2 applies (i.e. the contiguousness requirements), we are able to demonstrate two ways informativeness is inherited: If the parents are partially informative, the child will be as well. And if the input parent is very informative, and the output parent is partially informative, the child will be very informative. The first theorem follows from the No Dangling Nodes condition both being inherited and implying partial informativeness. The second goes beyond that: it relies on the fact that the very informative input parent has a path from each input to the membrane, no matter where the membrane falls; so if the output parent provides a crossover compatible output part, then it must pick up each one of those paths and connect them to an output.

To prove these theorems, we establish two lemmas. Lemma 4 points out that, if there is a path from an input to an output in some graph G, then that path must go through the membrane associated with any IO Partition of G. The intuition is straightforward: if the membrane is a fence between inputs from outputs, then a path from an input to an output must cross that fence. Formally:

Lemma 4 *If IOD Graph G satisfies the no dangling nodes condition and has at least one intermediate node, then for every path p from an input to an output, and for any IO Partition (Ψ, Ω), there exists some forward edge f in p such that $f \in F(\Psi, \Omega)$.*

Proof. Since there is a path from some input i to some input o, there must be some node n on the path in Ψ which connects to some node on the path n' in Ω (note, we are allowing $n = i$ or $n' = o$). Then edge (n, n') is a forward edge in $F(\Psi, \Omega)$. ■

Lemma 5 points out that no dangling nodes means that the forward links contained in any membrane must lie on a path from an input to an output. That is, if every step is a step on a path between an input and an output, then any step across the fence must be part of such a path.

Lemma 5 *If IOD Graph G satisfies the no dangling nodes condition and has at least one intermediate node, then for any IO Partition (Ψ, Ω), every forward edge $f \in F(\Psi, \Omega)$ must lie on a path from the input set to the output set and the set of forward edges $F(\Psi, \Omega)$ must be non-empty.*

Proof. By Lemma 4, $\exists f \in F(\Psi, \Omega)$. Now, f must connect two nodes n, n' in G. By the no dangling nodes condition, there must be a path $p = (i, \ldots, n, \ldots, o)$ and $q = (i', \ldots, n', \ldots, o')$ for $i, i' \in I$, $o, o' \in O$. Then we can construct the path $r = (i, \ldots, n, n', \ldots, o')$. f is on that path and connects the input set to the output set. ■

This is a critical component in the retention of informativeness, because if there were available forward links which led to blind alleys, then a path from an input could be directed into a blind alley, as seen in Figures 5 and 6a.

Now we are equipped to prove Theorem 3, which shows the inheritance of partial informativeness, and Theorem 4, which shows that a very informative input parent and partially informative output parent yield very informative children.

Theorem 3 *Let IOD Graphs G and G' be partially informative and satisfy the no dangling nodes condition. Then any crossover child produced by an input-contiguous IO partition of input parent G and an output-contiguous IO partion of output parent G' must be partially informative.*

Proof. By Theorem 2, the child C must satisfy the no dangling nodes condition. Since they are partially informative, G and G' each have at least one path from their input set to their output set, which we will name $p = (i, \ldots, o) \in N(G)$ and $p' = (i', \ldots, o') \in N(G')$. By Lemma 5, every edge in $M(\Psi, \Omega)$ is on a path

from I to O and every edge in $M(\Psi', \Omega')$ is on a path from I to O. Therefore, every edge in any crossover membrane $\hat{M}$ must be on a path from I to O. ■

Theorem 4 *Suppose input parent IOD Graph G is very informative and output parent G' is partially informative, and both satisfy the no dangling nodes condition. Then any crossover child produced by an input-contiguous IO partition of input parent G and an output-contiguous IO partion of output parent G' must be very informative.*

Proof. Since G is very informative, for each $i \in I$, $\exists$ a path $p(i, \ldots, o_i)$ for some $o_i \in O$. By Lemma 4, each path must have an edge in $M(\Psi, \Omega)$. By Lemma 5, every edge must lie on a path between the input set and the output set. Therefore, in C, every i must lie on a path to the output set. Therefore C is very informative. ■

However, the inheritance of full informativeness proves difficult to guarantee. In particular, the no dangling nodes condition and contiguous IO partitions are insufficient to assure the preservation of full informativeness. The intuition is as follows: Consider some input. Now, given full informativeness, there must be a path from that input to every output. However, a well-constructed crossover can direct all those paths to the *same* output. This breaks full informativeness. (However, these fully informative parents will yield a very informative child by Theorem 4.) Theorem 5 demonstrates how this remapping could occur:

Theorem 5 *Full informativeness of the parents is not necessarily preserved even if both parents satisfy the no dangling nodes condition and the IO partitions are contiguous.*

Proof. Suppose IOD Graphs G and G' are fully informative and satisfy the no dangling nodes condition with contiguous IO partitions (Ψ, Ω) and (Ψ', Ω'). Suppose they both have the name number of inputs J and number of outputs K and suppose $J = K$.

Suppose that all paths from inputs to outputs are unique and suppose they have no intermediate nodes in common. Name these paths $p(i, o)$ in G and $p'(i', o')$ in G'.

Then construct a crossover membrane $\hat{M}$ according to the following algorithm:

1. Choose, without replacement, $i \in I$, and $o' \in O'$.
2. For each $o \in O$ and $i' \in I'$, find the edge $e \in p(i, o)$ and $e' \in p'(i', o')$ such that $e \in M(\Psi, \Omega)$ and $e' \in M(\Psi', \Omega')$. Construct the edge e'' which connects the source of e with the destination of e'. Add e'' to $\hat{M}$.
3. Repeat until the sets I and O' have been exhausted.

In the implied child IOD Graph C, by construction, each $i \in I$ has $J = K$ paths to one output $o' \in O'$ and no other output. Therefore C is not fully informative. ■

6 Actionability

As described above, informativeness characterizes how information flowing into the system is utilized, in the sense that it characterizes how many inputs are eventually connected to an output. A symmetric concept is *actionability*, which characterizes the amount of informed behavior flowing out of a system, in the sense that it characterizes how many outputs are connected *from* an input. The levels of actionability mirror those of informativeness: A *partially actionable IOD Graph* has at least one path from an input to an output, a *very actionable IOD Graph* has a path from some input to every output, and a *fully actionable IOD Graph* has a path from every input to every output. On the other extreme, a *non-actionable IOD Graph* has no paths from inputs to outputs.

Obviously, actionability and informativeness are very closely related. Indeed, as is apparent from the definition, IOD Graphs are *non-informative* if and only if they are *non-actionable*, are *partially informative* if and only if they are *partially actionable*, and are *fully informative* if and only if they are *fully actionable*. They differ at the level of *very*, in that IOD Graphs can be *very informative* without being *very actionable* and vice versa.

The symmetry of these concepts allows the immediate extension of Theorems 3, 4, and 5 to actionability:

Theorem 6 *Let IOD Graphs G and G' be partially actionable and satisfy the no dangling nodes condition. Then any crossover child produced by an input-contiguous IO partition of input parent G and an output-contiguous IO partion of output parent G' must be partially actionable.*

Theorem 7 *Suppose input parent IOD Graph G is partially actionable and output parent G' is very actionable, and both satisfy the no dangling nodes condition. Then any crossover child produced by an input-contiguous IO partition of input parent G and an output-contiguous IO partion of output parent G' must be very actionable.*

Theorem 8 *The no dangling nodes condition and contiguous IO partitions are insufficient to assure the preservation of full actionability.*

These theorems are presented without proof.

Either actionability and informativeness could be more useful depending on the application. The main critereon would be which side (inputs or outputs) is more "expensive" and thus should be fully utilized. For example, in survey design in social science, collecting data for each variable is very expensive, so every input node should be fully utilized. In this case, informativeness is an important measure. Or, in robotics applications, motors/actuators are expensive, so every output node should be connected to and utilized by some inputs. Here, actionability is a more important measure.

7 Discussion

IOD Graphs are a broad class of graphs that can be used to solve problems, and this paper defines a crossover operation on these graphs which can be used for evolutionary computation. The utility of IOD Graphs rests on paths from inputs to outputs, so we define a measure of that connectedness through *informativeness* (and *actionability.*) We show some levels of informativeness will be preserved under crossover under certain conditions, the most important of which is the *no dangling nodes condition* which rules out blind alleys in the IOD Graph.

There are several reasons why fully informative IOD Graphs may not be optimal for particular problems. Here are three possibilities. First, as mentioned elsewhere, it may be costly to maintain connections. Second, in the context of e.g. a municipal water system, it may not be best for every input–some fresh water, some grey water–to connect to every output–some local waterways, some a water reclamation plant. Third, suppose the IOD Graph is a decision engine, in which each output triggers a specific action, all of which are known to be sometimes useful. Suppose there are a large number of inputs, many of which are known to be noise. In this case, it may be optimal to have a partially informative and very actionable IOD Graph so that noise is ignored but all actions are available.

One aspect of crossover in this framework is that crossover membranes are not unique; that is to say, given an input part Ψ and an output part Ω, there can, in general, be multiple crossover membranes which can yield a child. Moreover, these children might vary in their informativeness. This is known in the literature; Schaffer et al. (1992) named this the "competing conventions problem," in the sense that the internal meaning of nodes is contextual and dependent on the particular network structure, and there is nothing which forces that meaning to be consistent across networks. So two parents could have different "conventions" with regard to mappings of inputs to outputs which would undermine the effectiveness of the crossover child. Consider Figure 8, which depicts a competing conventions problem. Suppose the correct solution is a path from I1 to O1 and a path from I2 to O2. In this case, both parents solve the problem correctly while the child gets it exactly wrong. From a "competing conventions" perspective, this means that the "convention" or internal representation in nodes of these paths was not consistent/established across parents, leading to a breakdown during crossover.[11]

Along these lines, while crossover can result in a non-informative child from fully informative parents, it's also the case that the reverse is also possible. For example, Figure 9 depicts two non-informative parents with a fully informative child.

These two observations combined suggests that the selection of the crossover

[11] Note, there is also a crossover membrane which produces a child which gets the problem exactly right, namely the crossover membrane $\{(N1, N4), (N2, N3)\}$.

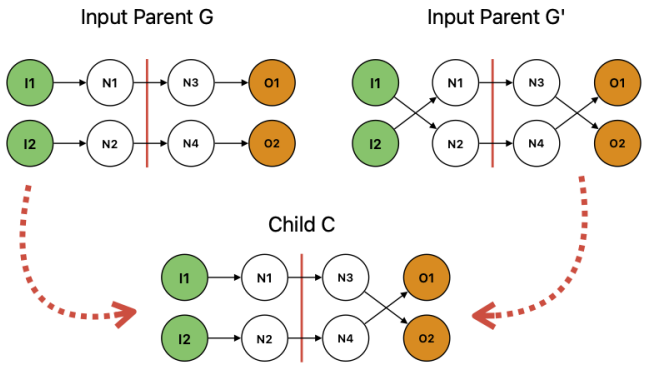


Figure 8: Crossovers may suffer from a competing conventions problem
Parents G and G' solve the problem, while the child C fails to solve it, because the parents "encode" the meaning of nodes differently.

membrane will be very important as we consider the implementations of in evolutionary computation. One possible algorithmic solution is to augment the crossover operation with endogenously "learned" crossover membranes or extend similarity-based methods such as Dragoni et al. (2014) to IOD Graphs. We intend to pursue this in future work.

The possibly profound implications of the competing conventions problem also suggests that traditional neural network updating (i.e. excluding crossover) might perform well precisely because, without crossover, there is a single internal "convention" of connections/weights that are encoded in the neural network at any particular point in time. There is no *competing* convention, which would make learning muddied.

In some applications, there are particular kinds of nodes that can or cannot be connected. For example, consider a municipal system of flow delivery pipes, which contain either water or natural gas. One would not want to consider crossovers which connect a water pipe to a natural gas receptor or vice versa. Crossover as defined here can be easily generalized to include this case; nodes could be 'tagged' and only connections between nodes of the same 'tag' could be considered in the crossover membranes.

Figure 10 depicts the distribution of informativeness as the degree density of the underlying IOD Graphs varies. In particular, we consider the universe of all IOD Graphs with 3 inputs, 2 outputs, 5 intermediate nodes, supposing that each input is connected to one node and each output is connected from one node. Then Figure 10 depicts, for each number of intermediate node connections, what fraction are Not, Partially, Very, and Fully informative. As we can see, non-informative graphs disappear quickly as degree density increases, but, on the other hand, very informative graphs persist to a fairly high level of degree density. If edges are somehow 'costly' to maintain, one might expect an

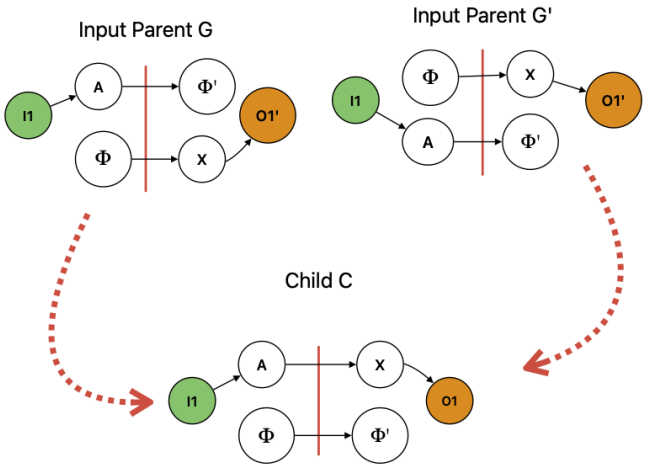


Figure 9: Two non-informative parents can have a fully-informative child
Parents G and G' have no path from their input to their output, while their Child C succeeds.

intermediate level of degree density, where partially and very informative graphs together dominate.[12]

Informativeness is only one way to measure the connectedness from inputs to outputs; in particular, one could consider metrics describing more subtle measures of connectedness, which count number of paths among possible paths, which could break down not, partial, very, and fully informative graph categories. The analysis of these metrics will be considered in future work.

8 Conclusion

Here we defined Input/Output Directed Graphs to capture a broad class of networks which connect inputs to outputs. This class includes feed-forward neural networks and perceptrons, electrical circuits, municipal water systems, chemical reaction networks, and data flow networks. Various studies have applied evolutionary methods to optimizing these networks in their distinct domains.

The goal of this study is to provide a strong theoretical foundation and common framework for crossover across all types of IOD Graphs and connect crossover as defined here with our proposed measure of informativeness (and actionability), which provides a measure of the connectedness of inputs to outputs in IOD Graphs.

[12]Note that the number of IOD Graphs for each number of edges varies widely, in particular, exponentially. That is, while there is exactly one graph with zero edges, and exactly one graph with 25 edges, there are over 3 million IOD Graphs with 13 edges.

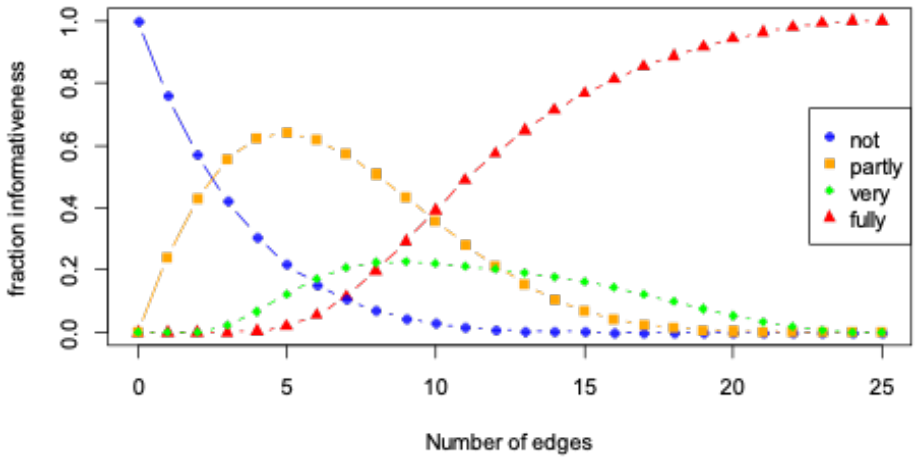
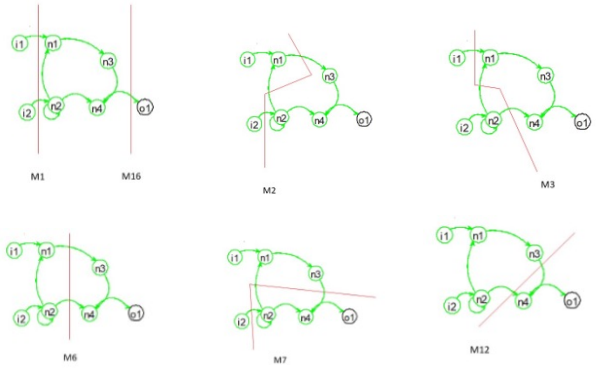


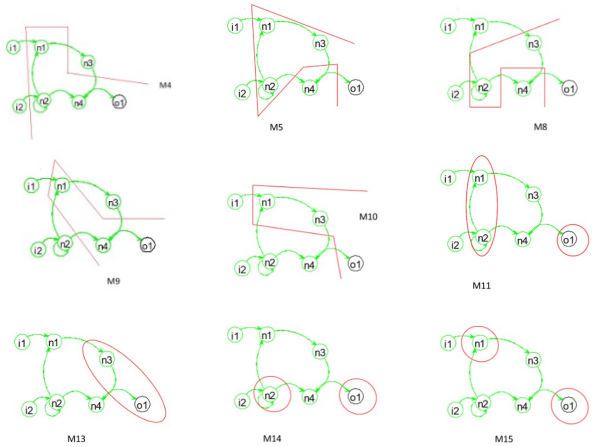
Figure 10: Distribution of IOD Graph Informativeness by Degree Density.
For low numbers of edges, most IOD Graphs are non-informative. As degree density increases, most become Partially and then Fully informative.

There are many directions for future work, e.g. generalized methods of constructing crossover membranes, results for IOD Graphs with nodes of different types or other parameters, generalization of informativeness to other types of graphs, and more nuanced, possibly continuous, measures of informativeness and actionability.

One area of future work is a computational implementation of crossover on general IOD Graph structures. Figure 11 shows output of one such computational implementation we are developing. In this Figure, all sixteen IO partitions of a particular IOD Graph are generated. We identify contiguousness and construct crossover membranes computationally. A general IOD Graph framework such as this will support the integration of a variety of methods and applications from different domains and support general evolutionary computation on IOD Graphs.



(a) Seven Contiguous IO Partitions



(b) Nine Non-contiguous IO Partitions

Figure 11: All IO Partitions of an example IOD Graph
This depicts all available partitions of an example IOD Graph. Most partitions are not contiguous.

References

Arifovic, J. and Gencay, R. (2001). Using genetic algorithms to select architecture of a feedforward artificial neural network. *Physica A: Statistical mechanics and its applications*, 289(3-4):574–594.

Bianchini, M., Gori, M., and Scarselli, F. (2005). Inside pagerank. *ACM Transactions on Internet Technology (TOIT)*, 5(1):92–128.

Braun, H. and Ragg, T. (1997). Enzo user manual and implementation guide, version 1.0.

Braun, H. and Weisbrod, J. (1993). Evolving neural feedforward networks. In *Artificial Neural Nets and Genetic Algorithms: Proceedings of the International Conference in Innsbruck, Austria, 1993*, pages 25–32. Springer.

Chandra, R., Frean, M., and Zhang, M. (2012). Crossover-based local search in cooperative co-evolutionary feedforward neural networks. *Applied Soft Computing*, 12(9):2924–2932.

Dragoni, M., Azzini, A., and Tettamanzi, A. G. (2014). Simba: A novel similarity-based crossover for neuro-evolution. *Neurocomputing*, 130:108–122.

D'Ambrosio, D. B., Gauci, J., and Stanley, K. O. (2014). Hyperneat: The first five years. *Growing Adaptive Machines: Combining Development and Learning in Artificial Neural Networks*, pages 159–185.

Elsken, T., Metzen, J. H., and Hutter, F. (2019). Neural architecture search: A survey. *Journal of Machine Learning Research*, 20(55):1–21.

García-Pedrajas, N., Ortiz-Boyer, D., and Hervás-Martínez, C. (2006). An alternative approach for neural network evolution with a genetic algorithm: Crossover by combinatorial optimization. *Neural Networks*, 19(4):514–528.

Gould, S. J. (1977). *Ontogeny and Phylogeny*. Harvard University Press.

He, C., Tan, H., Huang, S., and Cheng, R. (2021). Efficient evolutionary neural architecture search by modular inheritable crossover. *Swarm and Evolutionary Computation*, 64:100894.

Hodgkin, A. L. and Huxley, A. F. (1952). A quantitative description of membrane current and its application to conduction and excitation in nerve. *The Journal of physiology*, 117(4):500.

Ipsen, I. C. and Selee, T. M. (2008). Pagerank computation, with special attention to dangling nodes. *SIAM Journal on Matrix Analysis and Applications*, 29(4):1281–1296.

Jacob, F. (1977). Evolution and tinkering. *Science*, 196(4295):1161–1166.

Koza, J. R. (1999). *Genetic programming III: Darwinian invention and problem solving*, volume 3. Morgan Kaufmann.

Koza, J. R., Bennett, F. H., Andre, D., and Keane, M. A. (1997). Reuse, parameterized reuse, and hierarchical reuse of substructures in evolving electrical circuits using genetic programming. In *Evolvable Systems: From Biology to Hardware: First International Conference, ICES96 Tsukuba, Japan, October 7–8, 1996 Proceedings 1*, pages 312–326. Springer.

Kruschke, J. (1992). Alcove: an exemplar-based connectionist model of category learning. *Psychological Review*, 99(1):22.

Kurtz, K. J. (2007). The divergent autoencoder (DIVA) model of category learning. *Psychonomic Bulletin & Review*, 14(4):560–576.

Langville, A. N. and Meyer, C. D. (2006). A reordering for the pagerank problem. *SIAM Journal on Scientific Computing*, 27(6):2112–2120.

Lotf, J. J., Azgomi, M. A., and Dishabi, M. R. E. (2022). An improved influence maximization method for social networks based on genetic algorithm. *Physica A: Statistical Mechanics and its Applications*, 586:126480.

Maass, W. (1997). Networks of spiking neurons: the third generation of neural network models. *Neural networks*, 10(9):1659–1671.

Meng, X., Wong, S. H., Yuan, Y., and Lu, S. (2004). Characterizing flows in large wireless data networks. In *Proceedings of the 10th annual international conference on Mobile computing and networking*, pages 174–186.

Rosenblatt, F. (1957). *The perceptron, a perceiving and recognizing automaton Project Para*. Cornell Aeronautical Laboratory.

Sallinen, S., Luo, J., and Ripeanu, M. (2023). Real-time pagerank on dynamic graphs. In *Proceedings of the 32nd International Symposium on High-Performance Parallel and Distributed Computing*, pages 239–251.

Schaffer, J. D., Whitley, D., and Eshelman, L. J. (1992). Combinations of genetic algorithms and neural networks: A survey of the state of the art. In *[Proceedings] COGANN-92: International Workshop on Combinations of Genetic Algorithms and Neural Networks*, pages 1–37. IEEE.

Shanthi, A. and Parthasarathi, R. (2009). Practical and scalable evolution of digital circuits. *Applied Soft Computing*, 9(2):618–624.

Shepard, R., Hovland, C., and Jenkins, H. (1961). Learning and memorization of classifications. *Psychological Monographs*, 75:1–41.

Shinstine, D. S., Ahmed, I., and Lansey, K. E. (2002). Reliability/availability analysis of municipal water distribution networks: Case studies. *Journal of water resources planning and management*, 128(2):140–151.

Stanley, K. O., Clune, J., Lehman, J., and Miikkulainen, R. (2019). Designing neural networks through neuroevolution. *Nature Machine Intelligence*, 1(1):24–35.

Stanley, K. O., D'Ambrosio, D. B., and Gauci, J. (2009). A hypercube-based encoding for evolving large-scale neural networks. *Artificial life*, 15(2):185–212.

Stanley, K. O. and Miikkulainen, R. (2002). Evolving neural networks through augmenting topologies. *Evolutionary computation*, 10(2):99–127.

Sun, Y., Wang, H., Xue, B., Jin, Y., Yen, G. G., and Zhang, M. (2019a). Surrogate-assisted evolutionary deep learning using an end-to-end random forest-based performance predictor. *IEEE Transactions on Evolutionary Computation*, 24(2):350–364.

Sun, Y., Xue, B., Zhang, M., and Yen, G. G. (2019b). Completely automated cnn architecture design based on blocks. *IEEE transactions on neural networks and learning systems*, 31(4):1242–1254.

Topirceanu, A., Udrescu, M., and Vladutiu, M. (2015). Genetically optimized realistic social network topology inspired by facebook. In *Online Social Media Analysis and Visualization*, pages 163–179. Springer.

Unsleber, J. P. and Reiher, M. (2020). The exploration of chemical reaction networks. *Annual review of physical chemistry*, 71:121–142.

Uriot, T. and Izzo, D. (2020). Safe crossover of neural networks through neuron alignment. In *Proceedings of the 2020 Genetic and Evolutionary Computation Conference*, pages 435–443.

Wen, M., Spotte-Smith, E. W. C., Blau, S. M., McDermott, M. J., Krishnapriyan, A. S., and Persson, K. A. (2023). Chemical reaction networks and opportunities for machine learning. *Nature Computational Science*, 3(1):12–24.

Wu, Z. Y. and Clark, C. (2009). Evolving effective hydraulic model for municipal water systems. *Water resources management*, 23(1):117–136.

Zoph, B., Vasudevan, V., Shlens, J., and Le, Q. V. (2018). Learning transferable architectures for scalable image recognition. In *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 8697–8710.