# Evolutionary Learning of Graphical Causal Models from Behavior

H. Van Dyke Parunak[1[0000−0002−3434−5088]]

Parallax Advanced Research, Beavercreek, OH 45431 USA
van.parunak@{parallaxresearch.org, gmail.com}
https://parallaxresearch.org

**Abstract.** Many causal formalisms take the form of directed graphs, in which "causality" is implicitly defined as the relation between two nodes joined by an edge from the "cause" to the "effect," and modulated by parameters stored on the nodes (and sometimes on the agents that process the graph). Parunak has recently described a generalized formalism that embraces many varieties of this kind of structure, and also demonstrated the abilities, and limitations, of synthetic evolution to learn the required structures and parameters from specified behavior (sequence of states or actions) in one specific formalism. We review the generalized formalism, describe the specific variant in which evolutionary mechanisms have been explored, summarize the results obtained, and suggest how these methods may be extended to other causal formalisms.

**Keywords:** Graphical Causal Models · Agent-Based Models · Genetic Algorithms · Genetic Programming · Model Learning.

## 1 Introduction

"Behavior" in this paper refers to a trajectory of states or events exhibited by an agent or a system over time. A major problem in causal modeling is constructing a model to match specified behavior. Given a series of events and a formal modeling scheme, what configuration of the model best replicates the observed trajectory? Solutions to this problem are useful in two ways. Given observations of an unknown system's evolution, a fitted model could *forecast* its behavior in the future), while fitting a desired sequence of events to a familiar system could serve as a *planner and controller* to achieve that behavior.

The Bayesian foundations of much causal modeling invite fitting by regression over large volumes of data. But Bayesian methods have limitations compared with some other graphical models [14], and the amount of data required for reliable regression may not be available, particularly if the behavior of interest is represented by a single sequence rather than a large population of sequences.

This paper reports an approach based on synthetic evolution that has successfully been applied to a specific variety of graphical causal model (GCM), SCAMP (Social Causality using Agents with Multiple Perspectives). SCAMP includes a causal event graph (CEG) and a hierarchical goal network (HGN) [13,15,16]. Reference [14] develops a metamodel that compares SCAMP with a wide range of other graphical causal models. This paper examines the potential and limitations of the evolutionary approach on SCAMP models, and suggests how the more general metamodel would allow their extension to some other GCMs.

Section 2 reviews the metamodel from [14]. Section 3 summarizes SCAMP, the basis of the existing experiments on evolutionary learning. Section 4 summarizes this experience. Section 5 suggests how these methods can be applied to other GCMs that fit the metamodel.

## 2 A Formal Metamodel of Graphical Causal Models

GCMs represent causality as a directed graph. We avoid philosophical debates over the semantics of the directed edges, following Pearl: "cause" is an undefined primitive, like "point" and "line" in Euclidean geometry [20, pp. 27, 48]. In a GCM, we understand a directed edge as a causal claim, with a cause at the tail and an effect at the head.

A GCM requires not only a digraph with values associated with the nodes, but also an *updating mechanism* that updates the values on nodes. This mechanism is a significant component of the causal semantics of the formalism. Formally, [14] defines a graphical causal model

$$C \equiv \langle N, E, V, F, U \rangle \tag{1}$$

where the components are

- a set of nodes $N$;
- a set of directed edges between nodes $E \subset N \times N$;
- a set of values $V$ that nodes can carry;
- a function $F : N \to V$;
- an updating function $U : F \to F'$ that changes the values to which individual nodes evaluate.

$U$ often uses information associated with the edges to propagate causal effect from one node to another. Formalisms intended for human inspection and not computation may lack an explicit $V$, $F$, and $U$ and rely on informal, qualitative node updating, but do not evade this definition. The human users of such formalisms have an informal sense of the prominence of each node, and interpret the graph according to conventions that function as $U$.

In most methods with computational updating, $U$ is *analytic*, and involves *solving an equation*. [14] shows that an appropriately configured MAS can provide an *algorithmic* $U$.

Table 1 summarizes a wide range of GCMs, which are discussed in detail in [14]. It differs in two ways from the list in [14]:

- It distinguishes two components of the SCAMP causal graph: the causal event graph (CEG, L), whose nodes are event types, and the hierarchical goal network (HGN, M), whose nodes are states. Section 4.2 discusses the interaction of these components.
- It gives more detail on the three kinds of edges in the CEG: Agency edges (between successive events), Influence edges (indicating which nodes increase or decrease the probability of other nodes), and Resource edges. The Resource edges function like influence edges, except that they are activated by actual agent participation in events, rather than event probability (represented by pheromone deposits).

The various GCMs in Table 1 offer different opportunities for model learning.

- Informal models (A, B) are intended for human construction and interpretation, and so do not lend themselves to automated learning.
- Models whose edges are conditional probabilities, correlation coefficients, or weights (C-I) are usually fit with statistical regression methods [9,3], though these methods can be hampered by incomplete data and the curse of dimensionality.
- Model types (D-F) have no intrinsic notion of change through time, and so it is difficult to fit a behavioral trajectory to them. The strongest representations of time are in types (H, J-M).

Table 1: Common Causal Formalisms

| *1 Name* | *2 Nodes N* | *3 Values V* | *4 Edge Info E* | *5 Update U* |
|---|---|---|---|---|
| A. Factor Tree Analysis [7] | Informal | Informal | Informal | Informal |
| B. Causal Loop Diagrams [24] | Variables | Informal | Sign, delay | Informal |
| C. Path Diagram [26] | Variables | $\mathbb{R}$ | Correlation Coefficient | Multiplication of path coefficients |
| D. Causal Diagram [19] | Statements | [0,1] (belief) | Conditional Probabilities | Multiplication |
| E. Influence Net [23] | Events | [0,1] (occurrence) | Weighted Influences | Multiplication |
| F. Causal Influence Model [21] | Booleans, ordinals, categories | [0,1] | Weighted Influences | Multiplication |
| G. Influence Diagram [6] | Decisions Chance | Sets | Conditional Probabilities | Multiplication |
| H. POMDPs [8,22] | S(tates) A(ctions) O(bservations) | Sets | Conditional Probabilities | Dynamic Programming |
| I. Fuzzy Cognitive Map [10] | Concepts as fuzzy sets | [-1, 1] | Weight | Multiplication Thresholding |
| J. System Dynamics, Ordinary Differential Equations (ODEs) [2] | Variables | $\mathbb{R}$ | Equation | Integration |
| K. Stochastic Petri Net [4] | Variables Transitions | $\mathbb{R}+$ [0,1] | Sequence | Simulation |
| L. Causal Event Graph [14] | Event types | $[-1, 1]^n$ | Agency, Influence, Resource | Simulation |
| M. Hierarchical Goal Network [16] | States | $[0, 1]^4$ (Satisfaction, Frustration, Urgency, Toleration) | And, Or | Propagation of Min and Max values |

Little work has been done on using evolutionary methods to fit models.[1] This paper reports experience using evolutionary methods to fit a true GCM, SCAMP, to observed behaviors.

## 3 Agent Reasoning in SCAMP Models

The experiments described below involve two interacting subgraphs in a SCAMP model: a CEG and an HGN. This section summarizes the basic structure and operation of these components [14].

[1] [1], currently only in preprint, claims to do "evolutionary causal discovery," but in fact evolves an equation to map a set of independent variables to one dependent variable, and has no notion of causal direction.

SCAMP is stigmergic: agents belonging to different groups move through their environment, leaving marks and guiding their decisions by marks left by other agents. The main environment is the CEG, whose nodes are different types of events and whose edges indicate that an agent currently participating in one type of event could choose to participate in another. Each trajectory through the CEG represents a possible agent behavior.

Each CEG node has a feature vector with three kinds of dimensions. *Well-being* features are intrinsic consequences of participating in an event (e.g., impact on physical health; terrain difficulty). *Urgency* features capture the impact of participation on each group's goals. *Presence* features record the level of recent participation by agents of each group. An HGN for each group translates presence features on some nodes into their urgency features. The length of the feature vector is the number of well-being features plus twice the number of groups (one position for the urgency of the event to each group, the other for recent visits to the event by ghosts of each group). Each feature is constrained to $[-1, 1]$ .

Each agent carries a vector of preferences, each in $[-1, 1]$, defined over the same vector space in which features are defined. To choose its next action, the agent 1) computes the dot product of its preferences and the features of each alternative, 2) exponentiates each dot product to yield a positive number, 3) raises each to the power of a determinism parameter, and 4) normalizes them to form a roulette.

SCAMP's agents are polyagents [18]. Each actor has an avatar that sends out repeated waves of ghosts to explore possible futures. Each ghost considers the full feature vector of each alternative, and augments the presence feature of its avatar's group on the nodes that it visits. The avatar follows its group's presence features.

The SCAMP HGN [17] enhances TÆMS [5]. These systems propagate scalar values (in TÆMS, "quality") from the leaves of the network (connected to events in the real world) up the hierarchy to the root. The SCAMP HGN refines quality into four values in $[0, 1]$. Satisfaction and frustration reflect how strongly an event connected ("zipped") to a leaf goal supports or blocks the subgoal, and propagate up the net to the root. The root computes the urgency or frustration that the connected goal poses, which propagates back down. Figure 1 shows how an HGN operates.

1. The sum of presence features on each event node reflects the total recent participation in the event across all groups.
2. Each leaf subgoal in the HGN is zipped to one or more events. Support zips increase the subgoal's Satisfaction with increased event participation, while block zips increase its Frustration with increased participation.
3. Satisfaction and Frustration propagate up the HGN through *and* and *or* links. An *and* passes the minimum Satisfaction and the maximum Frustration of its lower-level subgoals up to the next higher goal, while multiple *or*s pass up the maximum Satisfaction and minimum Frustration of their subgoals.
4. At the root, $Urgency = 1 - Satisfaction$ and $Tolerance = 1 - Frustration$.
5. Urgency and Tolerance propagate back down. Support zips increment the urgency of their events by Urgency, while block zips increment it by $Tolerance1$.
6. Urgency features on the events then guide agent choices moving over the CEG.

## 4 Evolving SCAMP Models

As each agent moves through the CEG, it selects a specific trajectory that depends not only on the CEG's branching structure, but also on the agent's preference vector, which guides its choice among

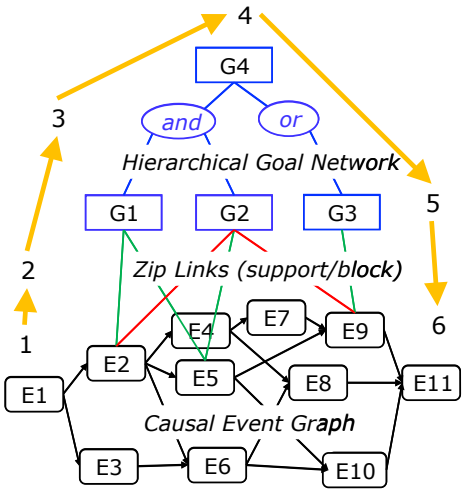


Fig. 1: Operation of SCAMP's HGN

alternative next events, and the HGN associated with its group. Having domain experts estimate preference vectors and construct HGNs is time-consuming and uncertain. The work summarized here is based on a use case in which the actual behavior of members of different groups (the sequence of event types in which they participate) can be observed, and we wish to recover preference vectors and HGNs that will lead an agent to replicate the observed trajectory, using synthetic evolution.

Evolutionary search defines a population of possible solutions, and repeatedly measures their fitness and modifying some of them for the next cycle. We must thus define three components:

1. The *genome* is the representation of a single solution as a string.
2. The *fitness function* measures how well a single genome solves the problem. In the work reported here, fitness is in $[0, 1]$, and lower fitness is better.
3. *Modification* changes less successful genomes in successive cycles, and may include one or two operations: *mutation* randomly changes one element of the genome, while *crossover*, which models sexual reproduction, splits two genomes and returns two genomes, each with the prefix of one of the parents and the suffix of the other.

A single run of a SCAMP model yields a *trajectory* for each agent, starting with the unique START node (E1 in Figure 1) and ending with the unique STOP node (E11). The agent's decisions are stochastic, so multiple runs may yield different trajectories. The *trajectory spectrum* of a set of runs is the list of trajectories that were sampled, with the number of times each was executed, and the *node spectrum* is the list of visited nodes with the number of visits for each. The *length* of a spectrum is the number of distinct trajectories (respectively, nodes) in the spectrum.

The evolutionary methods described here seek a genome that best reproduces a given *target trajectory*. In terms of Equation 1, the first method modifies the updating function $U$ of the CEG, while the second modifies the structure (nodes $N$ and edges $E$) of the HGN. We seek insights from these experiments that could be applied to evolutionary learning for models in Table 1. The focus of these experiments on a model's trajectory through time suggests that their most direct application will be to GCMs that describe a system's temporal evolution (H, J-M in Table 1).

Both sets of experiments use a randomly generated CEG with 300 nodes, being explored by two groups of agents with different preferences. The feature space has six elements: two for static

features defined by the modeler, two for an event's urgency to each group's goals, and two for the recent presence of each group on the node. The first set of experiments has no HGNs, and the first four elements are randomly assigned for each node. In the second, the first two are randomly assigned, and an evolved HGN adjusts the two urgency elements.

In both sets of experiments, the first group of agents is initialized with preferences randomly sampled around $[.8, -.8, .8, -.8, .8, -.8]$, while the second has preferences sampled around the inverse preferences $[-.8, .8, -.8, .8, -.8, .8]$. In each case, we summarize the genome, fitness function, and modification mechanisms used, describe the test data, and highlight results from the experiments that may be helpful in extending these methods to other GCMs.

### 4.1 Learning Agent Preferences [15]

A CEG's updating mechanism $U$ centers around the pheromones deposited by agent's ghosts on the presence features of nodes they visit. If HGNs are used, urgency features are also updated, but these are driven directly by presence features, and thus the focus returns to these features and the agent choices that generate them. Those choices, in turn, depend on each agent's *preference vector*. So the most straightforward way to match a target trajectory is to find a preference vector that will lead to that series of choices by the agents.

**Genome:** A natural genome for this problem is a candidate preference vector, which is represented as a Gray-coded bitstring.

**Fitness Function:** The fitness of a preference vector is the similarity of the trajectory it produces to the target trajectory. A natural candidate is the Levenshtein or string edit distance between the two trajectories, the minimum number of nodes in one trajectory that must be inserted, removed, or changed to transform it into the other ([12]), for which an $O(mn)$-time algorithm is available [25]. Lower values indicate fitter genomes.

Because each agent decision is based on a stochastic roulette, two different runs of a given genome against the reconstructed history of the feature vectors may yield different trajectories. The fitness of a genome is the total Levenshtein distance over 50 runs, divided by the number of perfect runs plus 1. Thus a fitness of 0 indicates a perfect match on every run. The algorithm stops when the fitness reaches 0.0, or at 500 generations.

**Modification:** To produce a new generation, each member of the current population is mutated (flipping each bit in the Gray-coded preference vector with probability 0.1), and crossed over with a partner that is at least as fit. This process triples the size of the population, so the resulting genomes are sorted by fitness and the resulting list is cropped back to the specified population size.

**Test Data:** Repeated runs of six agents, three from each group, with the initial preference vectors sampled around $[.8, -.8, .8, -.8, .8, -.8]$ and $[-.8, .8, -.8, .8, -.8, .8]$, yielded 131 complete trajectories, of which 12 were distinct, nine generated by G(roup) 0, and three different ones by G1. Variation across the trajectories results from 1) different initial preference vectors, due to sampling around the base value, and 2) the stochastic agent decisions at each step.

**Highlights of Results:** Each agent executes the evolutionary process for each trajectory that an agent completed, generating 131 evolutionary trials. 122 of these found a preference vector that successfully recovered the original trajectory, including nine of the 12 distinct trajectories originally observed. Eighty of these 122 were solved in only two generations, and all were solved in 12 generations or fewer.

Recovering a trajectory does not imply recovering the original preference vectors. Figure 2 shows the cosine similarity between the original and the evolved preference vectors, as a function

of trajectory length. Longer trajectories clearly require an evolved set of preferences to be more similar to the original, because there are more decision points at which the evolving agent might be led astray by divergent preferences.

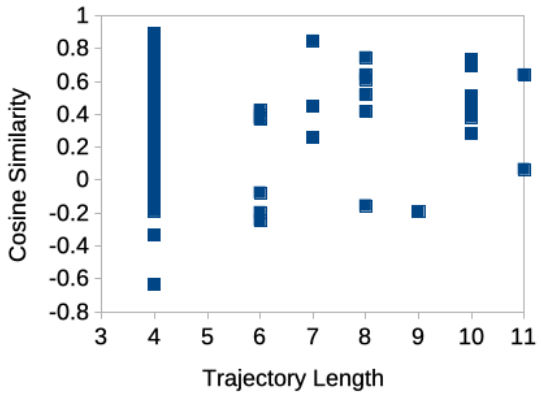


Fig. 2: Cosine similarity of original and evolved preference vectors

This set of experiments demonstrates the efficiency and effectiveness of fitting the numerical parameters of a particular kind of GCM (the SCAMP CEG) against behavioral data. It also emphasizes the importance of understanding the objective of the model. This approach is promising when the objective is to recover some model of a given form that will match observed behavior. If the objective is to replicate the actual parameters of the original model, the fitness function would need to focus on the parameters, not the generated behavior.

### 4.2 Learning Graph Connectivity [16]

Learning preference vectors demonstrates the ability to learn quantitative parameters of a model. GCMs have not only parameters, but (crucially) graphical structure. How well can evolution generate such structures? Reference [16] explores this potential using genetic programming (GP), a technique originally developed to evolve program trees for automatic software generation [11]. In the context of SCAMP, GP is applied to the HGN, a hierarchical network of goals that monitors the pheromone strength of a set of events and modulates their urgency features, thus affecting the choices of agents working through the CEG. Each leaf goal in the HGN is zipped to an event in the CEG that either *supports* or *blocks* the goal, increasing (respectively, decreasing) its level of *satisfaction*. Subgoals are combined into parent goals by either *and* (satisfaction of parent is the minimum satisfaction of all subgoals), or *or* (satisfaction of the parent is the maximum satisfaction of the subgoals). Figure 3 shows an HGN with zipped events inspired by the children's poem-story, "Little Miss Muffet." All of the events support their leaf goals, except for the event where the spider sits down beside Miss Muffet, blocking her goal to be secure.

**Genome:** The genome in this experiment is a preorder traversal of the HGN, as in Figure 4.

**Fitness Function:** Finding a good fitness function for evolving HGNs proved challenging. The string edit distance did not provide good guidance. Reference [16] formulates and tests five other

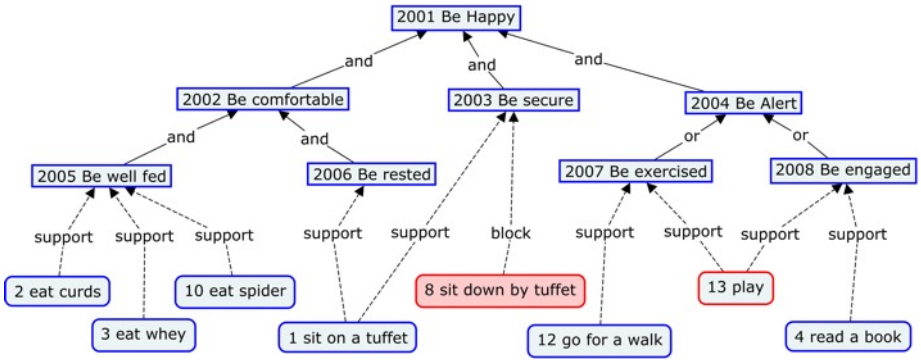


Fig. 3: A simple HGN. Rounded rectangles are CEG nodes.

```
Goal(and(
   Goal(and(
      Goal(support(E2), support(E3), support(E10)),
      Goal(support(E1))),
   Goal(support(E1), block(E8)),
   Goal(or(
      Goal(support(E12), support(E13)),
      Goal(support(E13), support(E4))))))
```

Fig. 4: Textual representation of an HGN

candidate fitness functions, all with a range of $[0, 1]$. The most effective (Fitness4) is based on the lengths of the target spectrum $|A|$, the test spectrum $|E|$, and their intersection $|A \cap E|$, subtracting from 1 so that lower is better:

$$Fitness4 \equiv 1 - (|A| \cap |E|)/|A| \tag{2}$$

In other words, a fitter HGN will visit more nodes in the target trajectory, and fewer nodes not in the HGN, than a less fit HGN.

**Modification:** In a genetic *algorithm* (Section 4.1), the genome can be treated as a bit string and mutation and crossover are straightforward. A genetic *program* must respect the structural integrity of the genome, making it more complicated.

Mutation walks through a preorder list of the nodes in the HGN. It mutates each node with a configurable probability (0.5 in the experiments reported here), replacing it with a new randomly generated node (which may differ from the original node in *and* vs. *or*). If the node being replaced is a leaf node, the new node grows a subtree. Otherwise it inherits the children of the replaced node.

Crossover first selects a partner, then selects a node from a preorder list of the receiving genome and replaces it (and thus its descendants) with a node (and descendants) from the partner.

Crossover can reduce the diversity of the population. This problem did not arise in evolving preferences, but was dramatic in evolving HGNs. The result is that the algorithm wastes time evaluating the fitness of identical genomes, while failing to explore as widely as it should. To solve this problem, while generating a new population, mutation or crossover of a selected member is repeated until it generates a genome not already in the set.

**Test Data:** The test data consists of trajectories generated by four different HGNs (to assess the similarity between the generating HGN and the recovered one), and one manually constructed trajectory (much longer than those generated by the HGNs).

HGN1 results from running the algorithm backwards to find an HGN whose trajectories differ maximally from the CEG with no HGN. These trajectories were very short (5-7 nodes), so a longer trajectory (8 nodes) was manually constructed. Evolution was unable to recover this complete trajectory, failing to include event E124. HGN2, HGN3, and HGN4 were manually designed to favor E124: HGN2 by discouraging events in the CEG that could bypass it (though the CEG still evaded E124), HGN3 by adding E124 as a supporting event to HGN2, and HGN4 consisting of a single goal zipped to E124 (yielding a trajectory spectrum identical to a CEG with no HGN).

**Highlights of Results:** HGN1 was easily recovered by all fitness functions but one, but with virtually no gain in fitness from the first to the last generation. No fitness function completely matched the manually constructed trajectory, but Fitness4 did the best, recovering 5 of the 8 event nodes, though with 3 false positives. It recovered trajectories generated by HGN2 and HGN4 perfectly, but reported two false positive notes on HGN3.

As with preference vectors, evolution can recover an HGN that generates the same trajectory as a furnished HGN, but not necessarily the original HGN.

The HGN experiments showed that it is possible to construct a GCM (specifically, an HGN) to produce an observed behavior (a sequence of events), but with two important qualifications.

1. A fitness function (string edit distance) that performed very well in fitting parameters floundered in fitting structure. Expect to spend resources exploring the space of possible functions.
2. Evolution could not find an HGN capable of guiding agents to a complete trajectory including E124, even though this event is on complete paths from START to STOP in the CEG. Apparently, internal dynamics, including not only the interaction of the HGN and the CEG but also the actions of agents in multiple groups, make this event inaccessible.

The point deserves further comment. The trajectory including E124 was generated by manual examination of the CEG. It never emerged from any execution of the model. In other words, it cannot be considered "observed behavior." However, a very reasonable application of these methods would be to find an HGN that would yield, not just an observed trajectory, but a desired one, and the case of E124 shows that in some cases, this may not be possible.

## 5 Applications to Other Graphical Formalisms

The work reported in Section 4 shows that evolutionary methods can learn central components of a specific kind of GCM (the preferences and goal structures of SCAMP's agents) from a specified behavioral trajectory. These components are qualitatively different: preference vectors are numeric arrays, while HGNs are graphical structures. Learning arrays proved much more straightforward than learning structures. In particular, the string edit distance between target and test trajectories served very well as a fitness function for evolving preferences, but not for structures.

Two temporal causal formalisms offer particular promise for evolving numeric vectors: the initial place markings of a stochastic Petri net (K), and the coefficients of the ODEs underlying a System Dynamics model (J). POMDPs (H), while temporal, rely on conditional probabilities, for which Bayes' Theorem and other formal tools offer a much more theoretically grounded updating approach. Still, evolution might be promising here (and in other models relying on conditional probabilities)

in estimating the sensitivity of different candidate variables, to get an initial conceptual model running before engaging in the extensive data collection needed to support these more formal methods. While non-temporal models by definition do not generate trajectories, one might treat a set of model outcomes as a pseudo-time series, and evolve a constant set of probabilities, along with values of root variables, to replicate that series. Synthetic evolution in this case is a model engineering tool to help the initial definition of a model's parameters.

Evolutionary discovery of the structure of a GCM is in principle applicable to any of the formal models in Table 1. The additional requirement beyond learning vectors is a formal grammar of the model structure, to ensure that initial generation of each genome and its modification by mutation and crossover yield correct structures. Again, temporal behavior is a useful input only in temporal models, but a sequence of anticipated model outcomes might serve the same function.

Evolution of SCAMP's HGN was feasible because the HGN was only a part of the system, and the underlying CEG could generate realistic behaviors even without it. Most GCMs do not have this bipartite graphical structure, and some part of the structure (at a minimum, input and output nodes) might need to be specified as a seed that evolution could then extend. A reasonable target for evolution to seek could be the concatenation of the vectors of input and output variables: find a model that, given this input, will produce this output.

An important observation from both sets of experiments is that matching *trajectories* may result from different *genomes*. SCAMP's causal dynamics, like those in living systems, pose a tension between agents' nature and nurture. An agent's behavior does not result simply from its internal structure and motivation, nor solely from its interactions with its environment (such as other agents). The two interact with and constrain one another. Experiments like these with other GCMs can help focus researchers' attention on the dynamics of their systems, as well as their parameters and structure, leading to better understanding and deployment.

Synthetic evolution has been described as "the second best way to solve any problem." Formally grounded approaches to model learning are always preferable, if the model structure and available data allow their application. But when the model structure does not meet the assumptions of existing formal methods, or when insufficient data is available, evolution holds promise. The experiments summarized in this paper demonstrate the applicability of these methods with models that do not fit the current probabilistic toolbox, and suggest that they can help bootstrap models with formal methods but lacking data, and perhaps even guide in data collection.

## References

1. Deng, O., Nishimura, S., Ogihara, A., Jin, Q.: Evolutionary causal discovery with relative impact stratification for interpretable data analysis (2024), `https://arxiv.org/abs/2404.16361`
2. Forrester, J.W.: Industrial Dynamics. MIT Press, Cambridge, MA (1961)
3. Glymour, C., Zhang, K., Spirtes, P.: Review of causal discovery methods based on graphical models. Frontiers in Genetics **10** (2019), `https://api.semanticscholar.org/CorpusID:173992893`
4. Haas, P.J.: Stochastic Petri Nets: Modelling, Stability, Simulation. Springer, New York (2002)
5. Horling, B., Lesser, V., Vincent, R., Wagner, T., Raja, A., Zhang, S., Decker, K., Garvey, A.: The TÆMS white paper (2004), `http://mas.cs.umass.edu/pub/paper_detail.php/182`
6. Howard, R.A., Matheson, J.E.: Influence diagrams. In: Howard, R.A., Matheson, J.E. (eds.) Readings on the Principles and Applications of Decision Analysis, vol. 2, pp. 719–762. Strategic Decisions Group, Menlo Park, CA (1984)
7. Ishikawa, K., Loftus, J.H.: Introduction to Quality Control. 3A Corporation (1990)

8. Kaelbling, L.P., Littman, M.L., Cassandra, A.R.: Planning and acting in partially observable stochastic domains. Artificial Intelligence **101**(1), 99–134 (1998). https://doi.org/https://doi.org/10.1016/S0004-3702(98)00023-X, http://www.sciencedirect.com/science/article/pii/S000437029800023X
9. Koller, D., Friedman, N.: Probabilistic Graphical Models: Principles and Techniques. Adaptive computation and machine learning, MIT Press (2009), https://books.google.co.in/books?id=7dzpHCHzNQ4C
10. Kosko, B.: Fuzzy cognitive maps. International Journal of Man-Machine Studies **24**(1), 65–75 (1986)
11. Koza, J.R.: Genetic Programming: On the Programming of Computers by Means of Natural Selection. MIT Press, Cambridge (1992), an extremely important extension of genetic algorithms, permitting the direct evolution of computer programs. The volume contains dozens of examples of genetic programming in many domains, including the evolution of programs to play PacMan, back up a semi trailer truck, balance a broom, and discover mathematical functions
12. Levenshtein, V.I.: Binary codes capable of correcting deletions, insertions, and reversals. Soviet Physics Doklady **10**(8), 707–710 (1966)
13. Parunak, H.V.D.: Social simulation for non-hackers. In: Van Dam, K.H., Verstaevel, N. (eds.) 22nd International Workshop on Multi-Agent-Based Simulation (MABS 2021). Springer (2021)
14. Parunak, H.V.D.: How to turn an MAS into a graphical causal model. Journal of Autonomous Agents and Multi-Agent Systems (2022). https://doi.org/https://doi.org/10.1007/s10458-022-09560-y
15. Parunak, H.V.D.: Learning actor preferences by evolution. In: Yang, Z., von Briesen, E. (eds.) The 2021 Conference of The Computational Social Science Society of the Americas. pp. 85–97. Springer Proceedings in Complexity, Springer International Publishing, Switzerland (2022)
16. Parunak, H.V.D.: Learning Agent Goal Structures by Evolution. Lecture Notes in Computer Science, Springer, Cham (2024)
17. Parunak, H.V.D., Belding, T., Bisson, R., Brueckner, S., Downs, E., Hilscher, R., Decker, K.: Stigmergic modeling of hierarchical task networks. In: Tosto, G.D., Parunak, H.V.D. (eds.) the Tenth International Workshop on Multi-Agent-Based Simulation (MABS 2009, at AAMAS 2009). vol. LNAI 5683, pp. 98–109. Springer (2009)
18. Parunak, H.V.D., Brueckner, S.: Concurrent Modeling of Alternative Worlds with Polyagents, pp. 128–141. LNCS, Springer, Berlin, Germany (2007)
19. Pearl, J.: Causality. Cambridge University Press, Cambridge, UK, 2 edn. (2009)
20. Pearl, J., Mackenzie, D.: The Book of Why. Basic Books, New York, NY (2018)
21. Pfautz, J., Cox, Z., Catto, G., Koelle, D., Campolongo, J., Roth, E.: User-centered methods for rapid creation and validation of Bayesian belief networks. In: Laskey, K.B., Mahoney, S.M., Goldsmith, J. (eds.) Fifth Bayesian Modeling Applications Workshop (UAI-AW 2007) at BMA '07. pp. 37–46 (2007)
22. Pynadath, D.V., Dilkina, B., Jeong, D.C., John, R.S., Marsella, S.C., Merchant, C., Miller, L.C., Read, S.J.: Disaster worlddecision-theoretic agents for simulating population responses to hurricanes. Computational & Mathematical Organization Theory **(forthcoming)** (2021)
23. Rosen, J.A., Smith, W.L.: Influence net modeling with causal strengths: An evolutionary approach. In: Command and Control Research and Technology Symposium (1996)
24. Sterman, J.: Business Dynamics. McGraw-Hill, New York, NY (2000)
25. Wagner, R.A., Fischer, M.J.: The string-to-string correction problem. Journal of the ACM **21**(1), 168âĂŞ173 (1974)
26. Wright, S.: The method of path coefficients. Ann. Math. Statist. **5**(3), 161–215 (1934)