Supplementary Material

Supplementary Material: On Automated Multi-objective Identification Using Grammar-based Genetic Programming

Dhruv Khandelwal, Maarten Schoukens, and Roland Tóth, *Member, IEEE,*

***Abstract*—This document contains the supplementary material for the contribution *On Automated Multi-objective Identification Using Grammar-based Genetic Programming*.**

***Index Terms*—System Identification, Tree Adjoining Grammar, Genetic Programming**

S1. Introduction

THE supplementary material presented in this document supports the concepts presented in the paper "On Automated Multi-objective Identification Using Grammar-based Genetic Programming". In particular, this document provides details required to ensure reproducibility of the ideas and results discussed in the aforementioned contribution. Hence, this document should be read in conjunction with the aforementioned contribution. To distinguish the elements of this document from that of the primary contribution, the letter 'S' is pre-fixed to references of sections, figures, algorithms, tables and equations in this document.

S2. Initialization

In this Section, we describe the initialization procedure for the Genetic Programming (GP) based identification method proposed in Sec. 5.

Recall that a Tree Adjoining Grammar (TAG) G is given by the tuple $\langle \mathfrak{N}, \mathfrak{T}, \mathcal{I}, \mathcal{A}, S \rangle$, where $\mathfrak{N}$ is the set of non-terminal labels, $\mathfrak{T}$ is the set of terminal labels, $\mathcal{I}$ is the set of initial trees, $\mathcal{A}$ is the set of auxiliary trees, and S is the label of the root node. An initial tree η is described by the tuple $\langle V, E, r \rangle$, where V is the set of vertices, E is the set of edges and r is the root-node. Similarly, an auxiliary tree α is described by the tuple $\langle V, E, r, f \rangle$, where f is the foot-node of the auxiliary tree. Finally, ϕ denotes the null set.

The algorithms used for initialization are presented in Alg. S1, S2 and S3. In the proposed initialization, each derivation tree ρ_i^0 in the initial population $X^{(0)}$ is grown upto a depth d that is randomly chosen from the range $[2, m_{\mathrm{id}}]$, where m_{id} is the maximum tree depth in the initial population, a hyper-parameter chosen by the user. The tree is initialized with an initial tree whose root node is labelled with start symbol S of grammar G, see Step 3 of Alg. S1. Throughout the initialization process, we maintain lists Π_{s} and Π_{a} to track leaves and internal vertices in the derived tree that can participate in a substitution and adjunction operation, respectively. If the chosen initial tree contains leaves available for substitution

Algorithm S1 The initialization scheme

Require: Grammar $G = \langle \mathfrak{N}, \mathfrak{T}, \mathcal{I}, \mathcal{A}, S \rangle$, maximum initial tree-depth m_{id}, population size n_{s}

1: Initialize $X^{(0)} \leftarrow \{\}$, $D^{(0)} \leftarrow \{\}$
2: **while** $i \leq n_{\mathrm{s}}$ **do** ▷ Iterate through each individual in the population
3: Select $\eta_i = \langle V, E, r \rangle \in \mathcal{I}$ randomly, with uniform distribution, such that $l(r) = S$
4: $\rho_i^0 \leftarrow \langle V_i, E_i, r_i \rangle$, where $V_i = \{r_i\}$, $E_i = \phi$, and $l(r_i) \leftarrow \eta_i$ ▷ Initialize the derivation tree
5: $\delta_i^0 \leftarrow \eta_i$ ▷ Initialize the derived tree
6: $\Pi_{\mathrm{s}} \leftarrow \{(\nu_k, \nu) \mid (\nu \in \Lambda(\delta_i^0)) \wedge (l(\nu) \in \mathfrak{N}) \wedge (\exists \eta = \langle V'', E'', r'' \rangle : l(\nu) = l(r''))\}$ ▷ List vertices in the derived tree that can be substituted to, and the corresponding vertex in the derivation tree
7: $(\rho_i^0, \delta_i^0) \leftarrow \textsc{substitute}(\rho_i^0, \delta_i^0, r_i, \Pi_{\mathrm{s}}, G)$
8: $l \leftarrow 1 + |\Pi_{\mathrm{s}}|$
9: $k \leftarrow 1$ ▷ A counter for derivation tree vertices
10: $\nu_k \leftarrow r_i$
11: Select $d \in [2, m_{\mathrm{id}}]$ randomly with uniform distribution
12: **while** $d_{\mathrm{t}}(\rho_i^0) \leq d$ **do** ▷ Grow derivation tree until chosen depth d
13: $\Pi_{\mathrm{a}} \leftarrow \{(\nu_k, \nu) \mid (\nu \in V) \wedge (\nu \notin \Lambda(\delta_i^0)) \wedge (\exists \alpha = \langle V', E', r', f' \rangle : l(\nu) = l(f'))\}$ ▷ List vertices in the derived tree that can be adjoined to, and the corresponding vertex in the derivation tree
14: $(\rho_i^0, \delta_i^0, \Pi_{\mathrm{a}}, V) \leftarrow \textsc{adjoin}(\rho_i^0, \delta_i^0, k + l, \Pi_{\mathrm{a}}, G)$
15: $k \leftarrow k + l$
16: $\Pi_{\mathrm{s}} \leftarrow \{(\nu_k, \nu) \mid (\nu \in \Lambda(\delta_i^0)) \wedge (l(\nu) \in \mathfrak{N}) \wedge (\exists \eta = \langle V'', E'', r'' \rangle \in \mathcal{I} : l(\nu) = l(r''))\}$ ▷ List vertices in the derived tree that can be substituted to, and the corresponding vertex in the derivation tree
17: $(\rho_i^0, \delta_i^0) \leftarrow \textsc{substitute}(\rho_i^0, \delta_i^0, \nu_k, \Pi_{\mathrm{s}}, G)$
18: $l \leftarrow 1 + |\Pi_{\mathrm{s}}|$
19: $\rho_i^0 \leftarrow \langle V_i, E_i, r_i \rangle$
20: $k \leftarrow k + 1$
return $X^{(0)} = \{\rho_i^0\}_{i=1}^{n_{\mathrm{s}}}, D^{(0)} = \{\delta_i^0\}_{i=1}^{n_{\mathrm{s}}}$

D. Khandelwal, M. Schoukens and R. Tóth are with the Control Systems Group, Department of Electrical Engineering, Eindhoven University of Technology, Eindhoven, The Netherlands, e-mail: D.Khandelwal@tue.nl

This research is supported by the Dutch Organization for Scientific Research (NWO, domain TTW, grant: 13852) which is partly funded by the Ministry of Economic Affairs of The Netherlands.

(listed in Π_s in Step 6), compatible[1] initial trees, chosen randomly with uniform distribution, are substituted in Step 7 of Alg. S1 and in Alg. S2. After any substitution or adjunction operation, list Π_a, consisting of all vertices in the derived tree that are available for adjunction, is updated. Each vertex listed in Π_s and Π_a is also paired with the corresponding node in the derivation tree that generates the vertex in the derived tree.

After initializing the derivation tree with an initial tree, along with the corresponding substituted trees, the derivation tree can be grown iteratively by adjoining auxiliary trees. In each iteration, a vertex available for adjunction is chosen randomly, with uniform distribution, from list Π_a (see Step 13 in Alg. S1). Subsequently, a compatible auxiliary tree is chosen randomly with uniform distribution and adjoined to the individual at the chosen location, see Step 14 in Alg. S1 and Alg. S3. Again, if required, suitable initial trees substituted to the newly adjoined tree in Steps 16 and 17 in Alg. S1. This process is repeated until the derivation tree has reached the required depth.

Algorithm S2 Substitute

Require: Derivation tree $\rho_i^j = \langle V, E, r\rangle$, derived tree δ_i^j, vertex $\nu_k \in V$, list of vertex pairs available for substitution Π_s, grammar G

1: $l \leftarrow 1$
2: **while** $\Pi_s \neq \phi$ **do**
3: Select any pair $(\nu_\rho, \nu_\delta) \in \Pi_s$
4: Select $\eta = \langle V', E', r'\rangle \in \mathcal{I}$ such that $l(\nu_\delta) = l(r')$
5: $V \leftarrow V \cup \{\nu_{k+l}\}$ with $l(\nu_{k+l}) = \eta$ ▷ Update the derivation tree
6: $E \leftarrow E \cup \{e\}$ where $e = \langle \nu_\rho, \nu_{k+l}\rangle$ and $g(e) = p_G(\nu_\rho, \rho_i^j)$
7: $\delta_i^j \leftarrow \delta_i^j[\nu_\delta, \eta]$ ▷ Update the derived tree
8: $l \leftarrow l + 1$
9: $\Pi_s \leftarrow \Pi_s \setminus \{(\nu_\rho, \nu_\delta)\}$
10: **return** ρ_i^j, δ_i^j

Algorithm S3 Adjoin

Require: Derivation tree $\rho_i^j = \langle V, E, r\rangle$, derived tree δ_i^j, index k for next vertex position in derivation tree, list of vertices pair available for adjunction Π_s, grammar G

1: Select a pair $(\nu_\rho, \nu_\delta) \in \Pi_a$ randomly with uniform distribution ▷ Select a vertex that can be adjoined to
2: Select $\alpha = \langle V', E', r', f'\rangle \in \mathcal{A}$ such that $l(\nu_\delta) = l(f')$. In case of multiple compatible auxiliary trees, select one randomly with uniform distribution.
3: $V \leftarrow V \cup \{\nu_k\}$ with $l(\nu_k) = \alpha$ ▷ Update the derivation tree
4: $E \leftarrow E \cup \{e\}$ where $e = \langle \nu_\rho, \nu_k\rangle$ and $g(e) = p_G(\nu_\rho, \rho_i^j)$
5: $\delta_i^j \leftarrow \delta_i^j[\![\nu_\delta, \alpha]\!]$ ▷ Update the derived tree
6: $\Pi_a \leftarrow \Pi_a \setminus \{(\nu_\rho, \nu_\delta)\}$ ▷ Update the list of vertices available for adjunction
7: **return** $\rho_i^j, \delta_i^j, \Pi_a, V'$

[1]See [1] for conditions under which a substitution operation is well-defined.

Variation operator	Hyper-parameter	Operator types	Illustration
Crossover	p_c	sub-tree	Fig. S1
Mutation	p_m	node insertion	Fig. S2
		branch insertion	Fig. S3
		node deletion	Fig. S4a
		branch deletion	Fig. S4b

TABLE SI: List of variation operators and their corresponding types.

S3. Variation Operators

In this Section, we describe the crossover and mutation operators employed in the proposed identification methodology. An overview of the operators used are listed in Tab. SI.

A. Crossover

In each iteration of the proposed method (see Alg. 1 of the paper), crossover and mutation operators are used to propose a new population. In the proposed algorithm, variation operators are performed on the derivation tree representation of each candidate solution. Recall that one of the main motivations for using TAG is to ensure that any model generated using a given TAG belongs the desired model set. In order to maintain this property during the evolutionary search, the crossover and mutation operators must be modified. The variation operators used in the proposed method are based on those used in [2], where the authors proposed crossover and mutation operators that ensure the resulting model structures also belong to the desired model set encoded in the TAG.

We employ a standard sub-tree crossover operator with additional constraints that ensure validity of the newly proposed adjunction operators. The crossover operator is applied to a pair of individuals in the population with probability p_c. Let ρ_1 and ρ_2 be two derivation trees selected for crossover and denote the corresponding derived trees as δ_1 and δ_2. The crossover operation consists of the following steps:

1) Select a vertex ν_1 in derivation tree ρ_1 randomly[2] and let ν_1' denote the parent of vertex ν_1. The label of vertex ν_1 denotes the auxiliary tree α_1 that is to be adjoined, in the derived tree representation δ_1, to the auxiliary tree[3] $\alpha_1' = \langle V_1, E_1, r_1, f_1\rangle$ represented by the label of the parent vertex ν_1'.
2) Select a vertex ν_2 and its parent vertex as ν_2' in derivation tree ρ_2 with corresponding auxiliary trees α_2 and $\alpha_2' = \langle V_2, E_2, r_2, f_2\rangle$, respectively, such that the following conditions are satisfied

$$\exists\ \nu_3 \in V_1 : \alpha_1'[\![\nu_3, \alpha_2]\!] \text{ is defined,} \tag{S1a}$$

$$\exists\ \nu_4 \in V_2 : \alpha_2'[\![\nu_4, \alpha_1]\!] \text{ is defined.} \tag{S1b}$$

[2]The vertex may be selected randomly with uniform distribution, however it is a convention to bias the random selection towards internal vertices, i.e., vertices that are not leaves. See [3].

[3]Note that the parent node label may correspond to an initial tree rather than auxiliary tree. For convenience, we assume that the parent node is an auxiliary tree, however the steps described here also apply for the case when the parent is an initial tree.

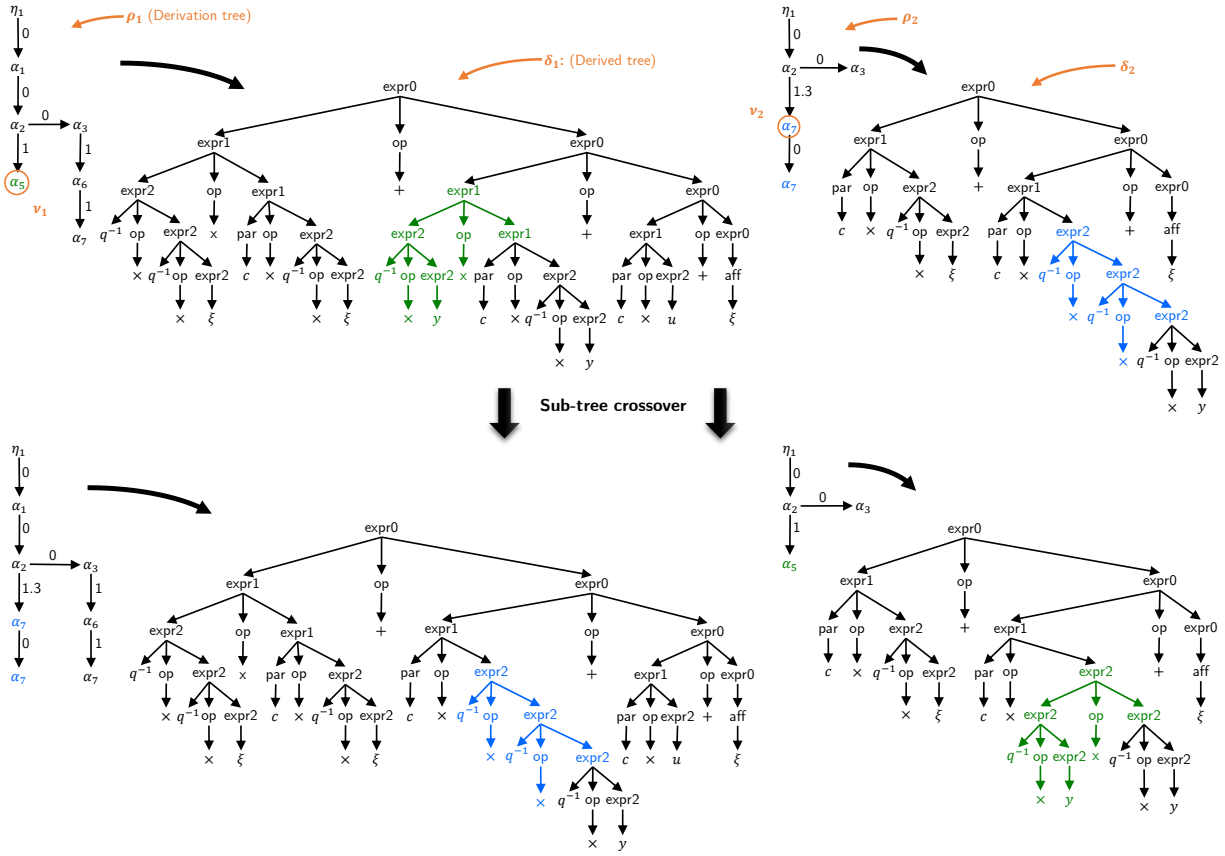


Fig. S1: The crossover operator illustrated on two individuals generated using G_{N}. The illustration depicts the two derivation trees ρ_1, ρ_2, the corresponding derived trees δ_1, δ_2 and vertices ν_1, ν_2 chosen for crossover. The coloured sections in the derivation trees and derived trees track the changes made in each individual.

In case of multiple possibilities, select one pair of vertices (ν_3, ν_4) randomly with uniform distribution. If there does not exist a pair (ν_3, ν_4) that satisfy (S1) for the chosen vertex ν_1 in derivation tree ρ_i, select another vertex ν_1 randomly and without replacement and repeat item 2.

3) Swap the sub-tree rooted at ν_1 in ρ_1 with the sub-tree rooted at ν_2 in ρ_2, and label the newly formed edges with the corresponding Gorn addresses

$$g(\langle \nu_1', \nu_2 \rangle) = p_{\mathrm{G}}(\nu_3, \alpha_1'), \tag{S2a}$$

$$g(\langle \nu_2', \nu_1 \rangle) = p_{\mathrm{G}}(\nu_4, \alpha_2'). \tag{S2b}$$

If the conditions in (S1) are not satisfied for any vertex ν_1 in ρ_1, then crossover cannot be performed for the two chosen individuals.

In Fig. S1 we illustrate the crossover operator on two individuals generated by TAG G_{N}. The models corresponding to the original individuals are the following.

$$y(k) = c_1 y^2(k-1) + c_2 u(k) + c_3 \xi(k-1)\xi(k-2) + \xi(k), \tag{S3}$$

$$y(k) = c_1 y(k-2) + c_2 \xi(k-1) + \xi(k). \tag{S4}$$

Subsequent to the crossover operator, the models corresponding to the new individuals are the following

$$y(k) = c_1 y(k-3) + c_2 u(k) + c_3 \xi(k-1)\xi(k-2) + \xi(k), \tag{S5}$$

$$y(k) = c_1 y^2(k-1) + c_2 \xi(k-1) + \xi(k). \tag{S6}$$

Remark S1. *Observe that, as illustrated in Fig. S1, the standard sub-tree crossover operator acting on the derivation tree representation is able to exchange components of the model expression (exponents and delays in the given example) that would not be possible for the same operator in the standard expression tree encoding used in GP. In fact, in an expression tree encoding of the model, exchange of delays or exponents would be achieved through a two-point crossover operation. This highlights an interplay between the representation of the model and the range of mapping achieved by the variation operator. In [4], the authors argue that this interplay determines, in part, the effectiveness of an Evolutionary Algorithm (EA) in finding the optimal solution.*

B. Mutation

In addition to the crossover operator, we also use the mutation operator to add variations to the proposed population. The mutation operator is applied to an individual in the population with a probability of p_m. The mutation operator

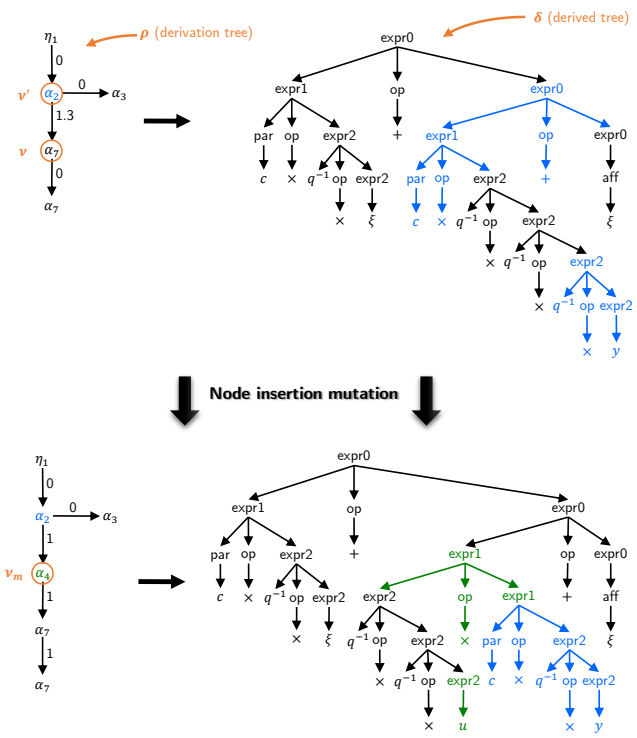


Fig. S2: The node insertion mutation operator illustrated on an individual generated using G_{N}. The illustration depicts the derivation tree ρ, the corresponding derived trees δ and vertex ν chosen for mutation. The coloured sections in the derivation trees and derived trees track the changes made in the individual. The new auxiliary tree (shaded in green) introduced into the individual is $\alpha_m = \alpha_4$.

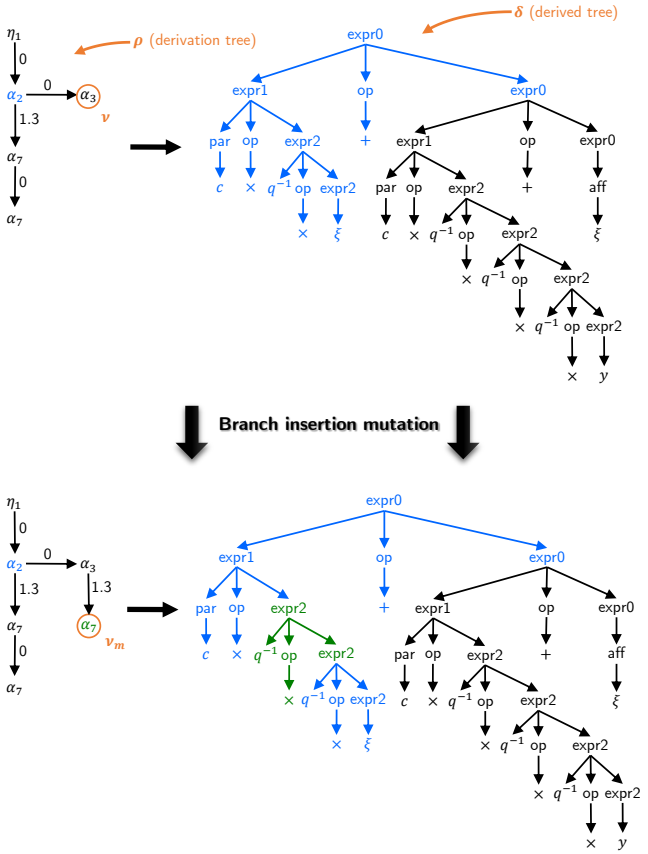


Fig. S3: The branch insertion mutation operator illustrated on an individual generated using G_{N}. The illustration depicts the derivation tree ρ, the corresponding derived trees δ and vertex ν chosen for mutation. The coloured sections in the derivation trees and derived trees track the changes made in the individual. The new auxiliary tree (shaded in green) introduced into the individual is $\alpha_m = \alpha_7$.

used in the proposed algorithm are of two types - insertion and deletion. Furthermore, there are two variations for each type of mutation operator - node or branch mutation. See overview in Tab. SI. For any chosen individual selected for mutation, the type of mutation operator used is determined randomly with uniform distribution[4].

The two variations of **insertion mutation** operator are described below.

1) Let the derivation tree of the individual selected for insertion mutation be denoted by ρ_1. Select a vertex ν in the derivation tree randomly with uniform distribution on all vertices except the root node, and denote the parent node of ν as ν'. Denote the auxiliary trees[3] corresponding to vertices ν and ν' in the derived tree representation as α and $\alpha' = \langle V', E', r', f' \rangle$, respectively. If node-type insertion is selected, go to Step 2, else if branch-type insertion is selected, go to Step 3.
2) **For node-type insertion operator:**
   a) select an auxiliary tree $\alpha_m = \langle V, E, r, f \rangle \in \mathcal{A}$ that satisfies the following conditions

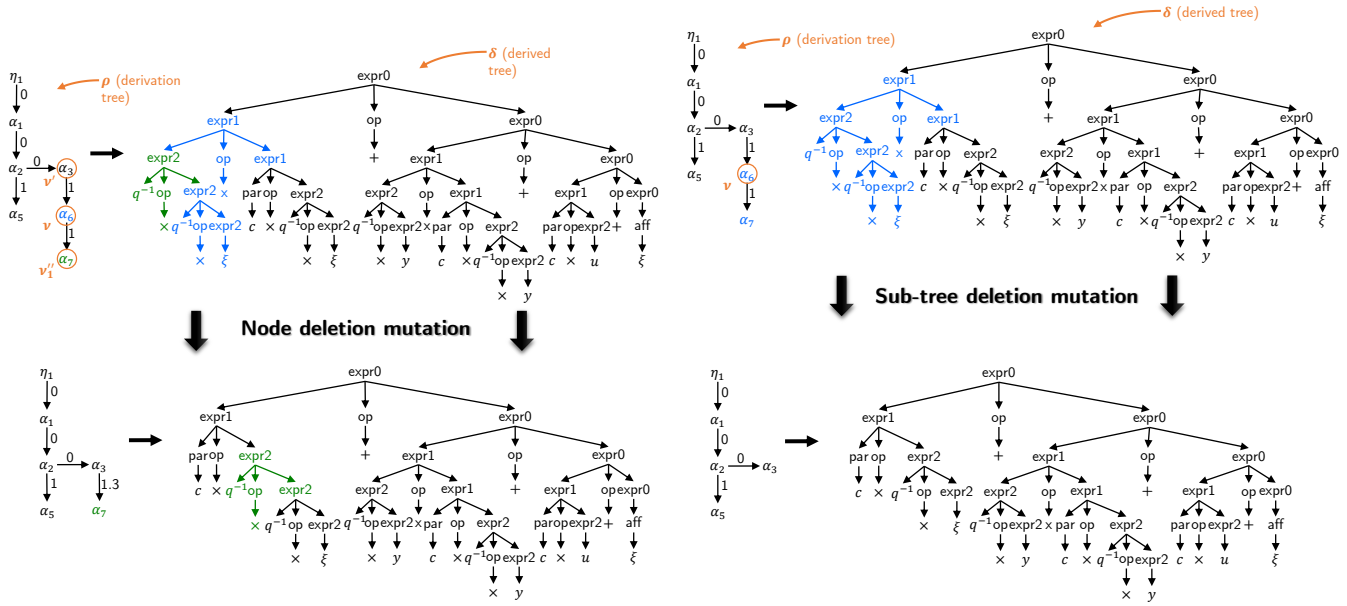
$$\exists\ \nu_1 \in V' : \alpha'[\![\nu_1, \alpha_m]\!] \text{ is defined,} \tag{S7a}$$

$$\exists\ \nu_2 \in V : \alpha_m[\![\nu_2, \alpha]\!] \text{ is defined.} \tag{S7b}$$

   For multiple choices of vertices pair (ν_1, ν_2), select one randomly with uniform distribution. If there exists no pair of vertices that satisfy the conditions in (S7), repeat Step 1 to choose a new vertex ν for mutation (without replacement).
   b) In ρ_1, delete edge $\langle \nu', \nu \rangle$ and insert a new vertex ν_m with label $l(\nu_m) = \alpha_m$. Make new edges $\langle \nu', \nu_m \rangle$ and $\langle \nu_m, \nu \rangle$ with the corresponding edge labels $g(\langle \nu', \nu_m \rangle) = p_{\mathrm{G}}(\nu_1, \alpha')$ and $g(\langle \nu_m, \nu \rangle) = p_{\mathrm{G}}(\nu_2, \alpha_m)$.
3) **For branch-type insertion operator:**
   a) Select an auxiliary tree $\alpha_m = \langle V, E, r, f \rangle \in \mathcal{A}$ that satisfies

$$\exists\ \nu_1 \in V : \alpha[\![\nu_1, \alpha_m]\!] \text{ is defined.} \tag{S8}$$

[4]Note that deletion operator cannot be used if the chosen derivation tree has tree depth less than 2. Similarly, insertion operator cannot be used if the derivation tree has depth equal to m_{d}.



(a) The node deletion mutation operator illustrated on an individual generated using G_{N}. The illustration depicts the derivation tree ρ, the corresponding derived trees δ and vertex ν chosen for mutation. The coloured sections in the derivation trees and derived trees track the changes made in the individual.

(b) The branch deletion mutation operator illustrated on an individual generated using G_{N}. The illustration depicts the derivation tree ρ, the corresponding derived trees δ and vertex ν chosen for mutation. The coloured sections in the derivation trees and derived trees track the changes made in the individual.

Fig. S4: Illustration of the node and sub-tree deletion mutation operators.

In the case of multiple choices for the pair (ν_1, α_m), select one pair randomly with uniform distribution. If there exists no pair (ν_1, α_m) that satisfies the conditions in (S8), repeat Step 1 to choose a new vertex ν for mutation (without replacement).

b) In ρ_1, insert a new vertex ν_m with label $l(\nu_m) = \alpha_m$. Make new edge $\langle \nu, \nu_m \rangle$ with edge label $g(\langle \nu, \nu_m \rangle) = p_{\mathrm{G}}(\nu_1, \alpha)$.

The two variations of the insertion mutation operator are illustrated in Fig. S2 and S3. In both examples, the model corresponding to the parent individual ρ is

$$y(k) = c_1 y(k-3) + c_2 \xi(k-1) + \xi(k). \tag{S9}$$

The model obtained as a result of node insertion mutation in Fig. S3 is

$$y(k) = c_1 u(k-2) y(k-1) + c_2 \xi(k-1) + \xi(k). \tag{S10}$$

In this example, the node insertion mutation introduces an input term $u(k)$ in the model expression, and moves the two delay-type auxiliary trees from the existing output term $y(k-3)$ to the newly introduced input-term.

The branch-insertion mutation operation in Fig. S3 results in the following model

$$y(k) = c_1 y(k-3) + c_2 \xi(k-2) + \xi(k). \tag{S11}$$

In this example, the mutation operator simply introduces a delay-type auxiliary tree to the existing noise-term $\xi(k-1)$.

The two variations of the **deletion operator** are described below.

1) Let ρ be the derivation tree chosen for the deletion operation. Select a vertex ν in the derivation tree randomly with uniform distribution on all vertices except the root node. Denote the parent node of ν as ν' and the auxiliary tree[3] generated by ν' in the derived tree representation as $\alpha' = \langle V', E', r', f' \rangle$. If the chosen variation of deletion operation is node-type, go to Step 2, else if the chosen variation is sub-tree type, go to Step 3.
2) **For node-type deletion operator:**
   a) Let the chosen vertex ν have n children vertices $\nu''_1, \ldots, \nu''_n$. Let the auxiliary trees generated by the children vertices be $\alpha''_1, \ldots, \alpha''_n$. The chosen vertex ν can be deleted if the auxiliary trees $\alpha''_1, \ldots, \alpha''_n$ can be adjoined to the auxiliary tree α'. This can be formalized as the following condition

$$\exists\, \{\nu_1, \ldots, \nu_n\} \subset V' : \alpha' [\![' \nu_1, \alpha''_1]\!] \ldots [\![\nu_n, \alpha''_n]\!] \text{ is defined.} \tag{S12}$$

   If there exist many choices for subset $\{\nu_1, \ldots, \nu_n\} \subset V'$ that satisfy (S12), select one randomly with uniform distribution. If there exists none, then vertex ν cannot be deleted. In that case, go to Step 1 to select another vertex ν in ρ without replacement.
   b) Delete vertex ν and edges connected to $\langle \nu, \nu''_1 \rangle, \ldots, \langle \nu, \nu''_n \rangle$. Make new edges $\langle \nu', \nu''_1 \rangle$, $\ldots$, $\langle \nu', \nu''_n \rangle$ with labels $g(\langle \nu', \nu''_1 \rangle) = p_{\mathrm{G}}(\nu_1, \alpha')$, $\ldots$, $g(\langle \nu', \nu''_n \rangle) = p_{\mathrm{G}}(\nu_n, \alpha')$.
3) **For sub-tree deletion operator:**
   a) Select a vertex ν in derivation tree ρ randomly with uniform distribution on all vertices except the root

Algorithm S4 Non-dominated sorting [5]

Require: Candidate solutions $\mathcal{M}^{(j)} = \{M_i^j(\theta_i^j)\}_{i=1}^{n_s}$

Initialize $l \leftarrow 1$
while $\mathcal{M}^{(j)} \neq \phi$ **do**
  $\mathcal{F}_l^{(j)} \leftarrow \phi$ ▷ Rank l solution front
  for each $M_i^j(\theta_i^j) \in \mathcal{M}^{(j)}$ **do**
    $\mathcal{F}_l^{(j)} \leftarrow \mathcal{F}_l^{(j)} \cup \{M_i^j(\theta_i^j)\}$ ▷ Include the individual temporarily in the solution front
    for each $M_q^j(\theta_q^j) \in \mathcal{F}_l \wedge M_q^j(\theta_q^j) \neq M_i^j(\theta_i^j)$ **do**
      if $M_i^j(\theta_i^j) \prec_J M_q^j(\theta_q^j)$ **then** ▷ If the newly added model dominates any other model in the solution front
        $\mathcal{F}_l^{(j)} \leftarrow \mathcal{F}_l^{(j)} \setminus \{M_q^j(\theta_q^j)\}$ ▷ remove the dominated model
      else if $M_q^j(\theta_q^j) \prec_J M_i^j(\theta_i^j)$ **then** ▷ If the newly added model is dominated by any other model in the solution front
        $\mathcal{F}_l^{(j)} \leftarrow \mathcal{F}_l^{(j)} \setminus \{M_i^j(\theta_i^j)\}$ ▷ Remove the newly added model
  $\mathcal{M}^{(j)} \leftarrow \mathcal{M}^{(j)} \setminus \mathcal{F}_l^{(j)}$ ▷ Remove all rank l models from the set of models yet to be sorted
  $l \leftarrow l + 1$
return $\mathcal{F}_1^{(j)}, \ldots, \mathcal{F}_{l-1}^{(j)}$

Algorithm S5 Crowding-distance based sorting [5]

Require: Solution front $\mathcal{F} = \{M_i(\theta_i)\}_{i=1}^{n}$, number of performance measures n_{obj}

Initialize $l \leftarrow 1$
for each $M_i(\theta_i) \in \mathcal{F}$ **do**
  Initialize $d(M_i(\theta_i) \leftarrow 0)$ ▷ Initialize the crowding distance of each model as 0
while $l \leq n_{\text{obj}}$ **do**
  $I = (i_1, \ldots, i_n) \leftarrow$ ORDERING$(\mathcal{F}, l)$ ▷ Compute the ordering of model indices that sorts the models based on performance measure J_l
  $d(M_{i_1}(\theta_{i_1})) \leftarrow \infty$ ▷ Set crowding distance on models on the ends of the frontier as infinite
  $d(M_{i_n}(\theta_{i_n})) \leftarrow \infty$
  for each $i \in \{i_2, \ldots, i_{n-1}\}$ **do**
    $d(M_i(\theta_i)) \leftarrow d(M_i(\theta_i)) + (J_l(M_{i+1}, \theta_{i+1}, \mathcal{D}_N) - J_l(M_{i-1}, \theta_{i-1}, \mathcal{D}_N))$ ▷ Compute the crowding distance of each individual as the sum of distances between the closest neighbours
  $l \leftarrow l + 1$
$\mathcal{F} \leftarrow$SORT$(\mathcal{F}, \{d(M_i(\theta_i))\}_{i=1}^{n})$ ▷ Sort the solution front based on crowding distance of the individuals
return $\mathcal{F}$

node.

b) Delete the sub-tree rooted at ν.

The two operations of deletion are illustrated in Fig. S4a and S4b. The model corresponding to the parent individual in both examples is

$$y(k) = c_1 u(k) + c_2 y^2(k-1) + c_3 \xi(k-1)\xi(k-2) + \xi(k). \quad \text{(S13)}$$

The result of the node deletion operator in Fig. S4a is

$$y(k) = c_1 u(k) + c_2 y^2(k-1) + c_3 \xi(k-2) + \xi(k). \quad \text{(S14)}$$

In this example, the mutation operator deletes one of the noise factors $\xi(k-2)$ and adjoins the delay-type tree to the other noise term $\xi(k-1)$ in the model expression. The model obtained due to branch deletion operator in Fig. S3 is

$$y(k) = c_1 u(k) + c_2 y^2(k-1) + c_3 \xi(k-1) + \xi(k). \quad \text{(S15)}$$

In this example, the sub-tree corresponding to $\xi(k-2)$ is completely deleted.

S4. Selection and Archiving

In the proposed algorithm, we employ the Pareto-based sorting and selection algorithm proposed in [5]. The selection mechanism relies on two ordering relations - Pareto-based ordering and crowding distance-based ordering. In the first step, the individuals of a population are sorted based on Pareto-dominance relationships. Computation of the solution fronts in a population can be performed using the Pareto-dominance based sorting algorithm presented in Alg. S4.

In the second step, the individuals of a population with the same rank are sorted based on crowding-distance relationships. Crowding distance is a measure of the density of solutions surrounding a given candidate model. In [5], the authors estimate the crowding distance of a model by measuring the largest cuboid that includes the given model in objective space such that no other model in the rank group is included in the cuboid. The crowding-based sorting algorithm is described in Alg. S5.

S5. Parameter Estimation for TAG G_{N}

In Sec. 5, we developed a methodology to solve the multi-criteria system identification problem in (3). The proposed methodology is fairly general and can be used to identify model structure automatically from measured data for an arbitrary TAG and arbitrary user performance measures. Due to the generality of the proposed methodology, global optimization techniques such as Particle swarm optimization (PSO) and Covariance matrix adaptation evolutionary strategies (CMA-ES) must be used to optimize model parameters for the different performance measures. However, for particular choices of TAG and performance measures, it is possible to use specialized parameter estimation algorithms that are computationally efficient.

In this Section, we restrict our scope to TAG G_{N} that generates models that belong to the polynomial Non-linear Auto-Regressive Moving-Average models with eXogenous inputs (P-NARMAX) class. Furthermore, for continuous objective functions in J_{cont} we consider the *one-step-ahead prediction error* and the simulation error. Under these restrictions, we discuss parameter estimation techniques for the two chosen performance measures.

A. Prediction Error Minimization

Any model structure M proposed by the GP algorithm using TAG $G_{\rm N}$ can be written as

$$y(k) = \sum_{i=1}^{p} \Bigg(c_i \prod_{j=0}^{n_u} u^{b_{i,j}}(k-j) \prod_{l=1}^{n_\xi} \xi^{d_{i,l}}(k-l) \prod_{m=1}^{n_y} y^{a_{i,m}}(k-m) \Bigg) + \xi(k). \quad \text{(S16)}$$

The one-step-ahead predicted output at time instant k, denoted by $\hat{y}(k \mid k-1)$, of a P-NARMAX model is given by the conditional expectation

$$\hat{y}(k \mid k-1) \coloneqq E_\xi[y(k) \mid \mathcal{D}_{k-1}, u_{\rm m}(k)], \quad \text{(S17)}$$

where, $E_\xi[\cdot]$ is the expectation operator with respect to the probability distribution of noise ξ. For a P-NARMAX model in the form of (S16), the one-step-ahead predictor can be formulated as the following non-linear filter

$$\hat{y}(k \mid k-1) = \sum_{i=1}^{p} \Bigg(c_i \prod_{j=0}^{n_u} u_{\rm m}^{b_{i,j}}(k-j) \prod_{l=1}^{n_\xi} \varepsilon_{\rm p}^{d_{i,l}}(k-l) \prod_{m=1}^{n_y} y_{\rm m}^{a_{i,m}}(k-m) \Bigg). \quad \text{(S18)}$$

where $\varepsilon_{\rm p}(k)$ is the one-step-ahead prediction error defined as

$$\varepsilon_{\rm p}(k) \coloneqq y_{\rm m}(k) - \hat{y}(k \mid k-1). \quad \text{(S19)}$$

Hence, if the initial conditions of the one-step-ahead predictor in (S18) are known, the predicted output $\{\hat{y}(k \mid k-1)\}_{i=n_{\rm l}+1}^{N}$, where $n_{\rm l} \coloneqq \max\{n_u, n_y, n_\xi\}$, can be computed recursively. If the initial conditions must be estimated and if the predictor model (S18) is asymptotically stable, then any transient errors introduced will asymptotically reduce to 0.

The sum-of-squares of the one-step-ahead prediction error for a given model $M(\theta)$ can be computed as

$$J_1(M, \theta, \mathcal{D}_N) = \frac{1}{N - n_{\rm l}} \sum_{k=n_{\rm l}+1}^{N} \left(y_{\rm m}(k) - \hat{y}(k \mid k-1)\right)^2. \quad \text{(S20)}$$

In the literature, several methods have been proposed to estimate model parameters in (S18) for the squared prediction error cost function J_1. For an overview, see [6, Chap. 3]. In this section, we introduce an iterative least squares based estimator to estimate model parameters in (S18) for objective function J_1, see [7] or [6, Chap. 3].

Let $\theta_{\rm p} = (c_1, \ldots, c_p)^\top$ be the parameter vector for the predictor model (S18). The predictor (S18) can be written in matrix form as

$$\hat{Y} = \phi_{\rm p} \theta_{\rm p} \quad \text{(S21)}$$

where $\hat{Y} = (\hat{y}(n_{\rm l}|n_{\rm l}-1), \ldots, \hat{y}(N_{\rm est}|N_{\rm est}-1))^\top$ is the vector of one-step-ahead predicted outputs, $\phi_{\rm p} = (\varphi_{{\rm p},1}, \ldots, \varphi_{{\rm p},N_{\rm est}})^\top$ is the regression matrix with $k^{\rm th}$ row being

$$\varphi_{{\rm p},k} = \Big(\prod_{j=0}^{n_u} u_{\rm m}(k-j)^{b_{1,j}} \prod_{l=1}^{n_\xi} \varepsilon_{\rm p}(k-l)^{d_{1,l}} \prod_{m=1}^{n_y} y_{\rm m}(k-m)^{a_{1,m}}, \ldots, \prod_{j=0}^{n_u} u_{\rm m}(k-j)^{b_{p,j}} \prod_{l=1}^{n_\xi} \varepsilon_{\rm p}(k-l)^{d_{p,l}} \prod_{m=1}^{n_y} y_{\rm m}(k-m)^{a_{p,m}} \Big). \quad \text{(S22)}$$

The regression matrix $\phi_{\rm p}$ is a function of the unknown prediction error $\varepsilon_{\rm p}$. Hence, to estimate the model parameters in (S16), we use an identification procedure that iteratively estimates model parameters $\theta_{\rm p}$ and prediction error $\varepsilon_{\rm p}$.

Reformulate the predictor model in (S18) as

$$\hat{y}(k \mid k-1) = f_{uy}(u_{\rm m}(k-1), \ldots, u_{\rm m}(k-n_u), y_{\rm m}(k-1), \ldots, y_{\rm m}(k-n_y)) + f_{uy\xi}(u_{\rm m}(k-1), \ldots, u_{\rm m}(k-n_u), y_{\rm m}(k-1), \ldots, y_{\rm m}(k-n_y), \varepsilon_{\rm p}(k-1), \ldots, \varepsilon_{\rm p}(k-n_\xi)), \quad \text{(S23)}$$

where f_{uy} is the part of the model expression without any noise terms and $f_{uy\xi}$ is the part of the model with noise terms. In order to obtain an initial estimate of the prediction error sequence, we estimate model parameters for the underlying polynomial Non-linear Auto-Regressive models with eXogenous inputs (P-NARX) predictor model that can be computed by ignoring $f_{uy\xi}$ in (S23). The P-NARX predictor model can also be written in matrix form

$$\hat{Y} = \phi_{uy} \theta_{uy}, \quad \text{(S24)}$$

where the definitions of ϕ_{uy} and θ_{uy} is similar to the P-NARMAX case. The minimizer of J_1 for the predictor model in (S24) can be computed as the least squares estimate $\hat{\theta}_{uy}$ of θ_{uy}, and is computed as

$$\hat{\theta}_{uy} = \left(\phi_{uy}^\top \phi_{uy}\right)^{-1} \phi_{uy}^\top \hat{Y}. \quad \text{(S25)}$$

Using the estimate $\hat{\theta}_{uy}$, the initial estimate of the prediction error $\hat{\varepsilon}_{\rm p}^{(0)}$ can be computed as

$$\begin{aligned} \hat{\varepsilon}_{\rm p}^{(0)}(k) &= y_{\rm m}(k) - \hat{y}(k|k-1), \\ &= y_{\rm m}(k) - \varphi_{uy,k} \hat{\theta}_{uy}, \end{aligned} \quad \text{(S26)}$$

where $\varphi_{uy,k}$ is the $k^{\rm th}$ row of the regressor matrix.

The initial estimate of prediction error $\hat{\varepsilon}_{\rm p}^{(0)}$ can be used to begin the iterative estimation procedure. Let h be the iteration index. In each iteration, the parameter estimate $\hat{\theta}_{\rm p}^{(h)}$ can be computed as the linear least squares estimate for the predictor model in (S21) where the prediction error terms in the regressor matrix $\psi_{\rm p}$ is replaced by the estimate $\hat{\varepsilon}_{\rm p}^{(h-1)}$. Based on the new parameter estimate, the prediction error estimate can be updated as

$$\begin{aligned} \hat{\varepsilon}_{\rm p}^{(h)}(k) &= y_{\rm m}(k) - \hat{y}(k|k-1), \\ &= y_{\rm m}(k) - \varphi_{{\rm p},k} \hat{\theta}_{\rm p}^{(h)}. \end{aligned} \quad \text{(S27)}$$

The iterations are stopped when the prediction error (or parameter) estimates converge within some user-specified tolerance level ϵ:

$$\sum_{k=1}^{N_{\rm est}} \left(\hat{\varepsilon}_{\rm p}^{(h)}(k) - \hat{\varepsilon}_{\rm p}^{(h-1)}(k) \right)^2 < \epsilon. \quad \text{(S28)}$$

The final estimate $\hat{\theta}_{\rm p}$ minimizes the cost function J_1 for the grammar $G_{\rm N}$. The estimate can be computed efficiently using a sequence of least squares estimators, and typically converges in a small number of iterations (typically less than 10 iterations, see [6]). If the lower-level of the original bi-level problem in (2) is simplified as in (3), then $\hat{\theta}_{\rm p}$ is one of the parameter estimates in the feasible set of the parameter space.

B. Simulation Error Minimization

1) The simulation model: A simulation model of (S16) can be defined as the conditional expectation of the output $y(k)$ with respect to the distribution of the noise, i.e.,

$$y_{\rm s}(k) \coloneqq E_{\xi}[y(k)]. \tag{S29}$$

The simulation response $y_{\rm s}(k)$ in (S29) represents the deterministic response of model (S16). By comparing (S29) and (S17), we observe that the simulation response can also be interpreted as an infinite-step-ahead prediction model. This effectively negates the auto-regressive components of a prediction model.

The sum-of-squares of the simulation error $\varepsilon_{\rm s}(k) \coloneqq y_{\rm m}(k) - y_{\rm s}(k)$ for a given model $M(\theta)$ can be computed as

$$J_2(M, \theta, \mathcal{D}_N) = \frac{1}{N - n_{\rm l}} \sum_{k=n_{\rm l}+1}^{N} \left(y_{\rm m}(k) - y_{\rm s}(k)\right)^2. \tag{S30}$$

For a given stochastic model in the form of (S16), the computation of simulated output as per (S29) is not trivial. In [8], the authors demonstrated that, in general, the simulation model of a finite-order P-NARMAX model may be given by an Non-linear Infinite Impulse Response (NIIR) model, thereby necessitating a approximation. The authors in [8] proposed the so-called *$l-$approximation simulation model* for a P-NARMAX model, where l is a parameter that determines the accuracy of the simulation model approximation. For brevity, we reproduce only the main result of [8] here, interested readers are referred to [8] for more details. Under the assumption that ξ is independent of u and that $\xi(k) \sim \mathcal{N}(0, 1)$, the $l-$approximate simulation model of (S16) is given by

$$y_{{\rm s},l}(k) = \sum_{i \in P_{\rm e}} \Bigg(c_i \prod_{j=0}^{n_u+l} u^{b_{i,j}}(k-j) \prod_{r=1}^{n_y} y_{{\rm s},l}^{a_{i,r}}(k-l-r) \prod_{q=1}^{n_\xi+l} (d_{i,q}-1)!! \Bigg), \tag{S31}$$

where $P_{\rm e} \coloneqq \{i \in [1, p] \mid d_{i,q} \text{ is even } \forall q \in [1, n_\xi]\}$ and $(n-1)!! \coloneqq \frac{n!}{\frac{n}{2}! 2^{\frac{n}{2}}}$.

In comparison with prediction error minimization, simulation-error-based estimation of P-NARMAX models is a far more challenging non-convex optimization problem. While the predictor model in (S18) is formulated in terms of known past measured input and output, the simulation model in (S31) is formulated in terms of the unknown past simulation output. This leads to complex non-linear dependencies between the parameters to be estimated and the simulation output. In the following, we present an optimization method based on Multi-Step Prediction (MSP) [9] to solve the non-convex simulation error minimization problem for the class of P-NARMAX models. Unlike [9], the MSP algorithm presented in this section is applicable to the entire class of P-NARMAX models due to the $l-$approximate simulation concept introduced in [8].

2) The MSP algorithm: The basic concept of the MSP algorithm is as follows. Let the $l-$approximate simulation model be parameterized by parameter vector $\theta_{{\rm s},l}$. Define, for some $\tau \in \mathbb{Z}_{>0}$, such that $\tau \geq l$, the τ-step-ahead predictor of the $l-$approximate simulation model (S31), as follows

$$\hat{y}(k|k-\tau) = \hat{f}_\tau \left(U_{{\rm m}0}^{n_u+\tau}(k), Y_{{\rm m}\tau+1}^{n_y+\tau}(k) \right), \tag{S32}$$

where, $\hat{y}(k|k-\tau)$ is the τ-step-ahead prediction, $\hat{f}_\tau$ is the τ-set-ahead predictor (computational aspects will be discussed in the sequel), $U_{{\rm m}0}^{n_u+\tau}(k) = \{u_{\rm m}(k-i)\}_{i=0}^{n_u+\tau}$ and $Y_{{\rm m}\tau+1}^{n_y+\tau}(k) = \{y_{\rm m}(k-i)\}_{i=\tau+1}^{n_y+\tau}$ are the set of past measured inputs and outputs, respectively.

Remark S2. *The predictor* (S32) *is the τ-step-ahead predictor of the l-approximate simulation response in* (S31) *and should not be confused with the predictor of the P-NARMAX model in* (S16)*.*

The predictor (S32) allows us to approximate $l-$approximate simulation (S31) using measured data. As $\tau \to \infty$, the τ-step-ahead prediction output approximates arbitrarily well the $l-$approximate simulation response $y_{\rm s}(l)$, i.e.,

$$\lim_{\tau \to \infty} \hat{y}(k|k-\tau) = y_{{\rm s},l}(k). \tag{S33}$$

Hence, in the MSP algorithm, the simulation model parameters $\theta_{{\rm s},l}$ is approximated by a sequence of parameter estimates $(\hat{\theta}_{{\rm s},l})_\tau$ for $\tau = l, l+1, ...\infty$, that optimize the cost function

$$(\hat{\theta}_{{\rm s},l})_\tau^* = \arg\min J_\tau(\theta_{{\rm s},l}), \tag{S34}$$

where $J_\tau(\theta_{{\rm s},l})$ is the 2-norm of the squared τ-step-ahead prediction error cost function. In practice, a stopping criteria based on convergence of parameter estimates or maximum number of iterations is used.

Remark S3. *In principle, the prediction model* (S32) *can be computed by recursively replacing the output terms model expression in* (S31)*, and re-formulating the subsequent model as a non-linear filter in terms of measured inputs and outputs. However, since the prediction model in* (S32) *is polynomial, performing multiple recursive substitutions of the output terms may result in polynomial growth of the order and the length of the model, making it infeasible to compute for large values of τ. A feasible approach for computing the τ-step-ahead predictor is discussed in the following subsection.*

Remark S4. *The role of parameter τ in the MSP algorithm is similar to the role of parameter l in the $l-$approximation method for computing the simulation model. The crucial difference between the two is that the $l-$approximation scheme determines the accuracy of a* simulation model*, while parameter τ ultimately determines the accuracy of a* prediction model*.*

In the following, we discuss the computational structure of the MSP algorithm. For $\tau = l$, the predictor model of (S31) can be written in matrix form as

$$\hat{Y}_l(\theta_{\mathrm{s},l}) = \phi_l \theta_{\mathrm{s},l}, \tag{S35}$$

where $\hat{Y}_l = (\hat{y}(l+1|1), \ldots, \hat{y}(N_{\mathrm{est}}|N_{\mathrm{est}} - l))^\top$ is the vector of l-step-ahead predicted outputs, $\phi_l = (\varphi_l(l+1), \ldots, \varphi_l(N_{\mathrm{est}}))^\top$ is the regression matrix. Rows $\varphi_l(k)$ consist of monomials in u_{m} and y_{m} and can be derived from (S31). Computation of ϕ_τ is discussed in detail in the following section. Similarly, the matrix-form of (S32) for $\tau > l$ can be computed by recursively substituting output terms in (S32), and can be written as

$$\hat{Y}_\tau(\theta_{\mathrm{s},l}) = \phi_\tau \vartheta_\tau(\theta_{\mathrm{s},l}), \tag{S36}$$

where $\vartheta_\tau(\theta_{\mathrm{s},l})$ is the non-linear parameterization (due to the recursive substitutions of the output terms) of the τ-step-ahead predictor model with respect to (w.r.t) $l-$approximate simulation model parameters $\theta_{\mathrm{s},l}$, and ϕ_τ is the corresponding regressor matrix.

For a given τ, the optimization problem (S34) can written as

$$(\hat{\theta}_{\mathrm{s},l})_\tau^* = \arg\min \frac{1}{N_{\mathrm{est}} - \tau} (Y_{\mathrm{m}}(\tau) - \hat{Y}_\tau(\theta_{\mathrm{s},l}))^\top (Y_{\mathrm{m}}(\tau) - \hat{Y}_\tau(\theta_{\mathrm{s},l})), \tag{S37}$$

where $Y_{\mathrm{m}}(\tau) = (y_{\mathrm{m}}(\tau+1), \ldots, y_{\mathrm{m}}(N_{\mathrm{est}}))^\top$ is a vector of measured outputs of the estimation dataset $\mathcal{D}_{\mathrm{est}}$. Gauss-Newton method is used to solve the non-linear least squares problem in (S37).

The MSP algorithm consists of two loops - the outer loop iterates over prediction time-horizon τ, while the inner loop corresponds to Gauss-Newton iterations required for solving the optimization problem in (S37) for a fixed time horizon τ. Denote the parameter estimate for the i^{th} Gauss-Newton iteration corresponding to time horizon τ as $(\hat{\theta}_{\mathrm{s},l})_\tau^i$. The algorithm is initialized with the linear least squares estimate $(\hat{\theta}_{\mathrm{s},l})_l^*$ (see (S35)). Set $\tau = l+1$ and $(\hat{\theta}_{\mathrm{s},l})_\tau^0 = (\hat{\theta}_{\mathrm{s},l})_{\tau-1}^*$. For each time horizon τ, the Jacobian is given by

$$\begin{aligned}\nabla_{\theta_{\mathrm{s},l}} J_\tau(\theta_{\mathrm{s},l}) &= \frac{2}{N_{\mathrm{est}} - \tau} \left(\hat{Y}_\tau(\theta_{\mathrm{s},l})^\top - Y_{\mathrm{m}}(\tau)^\top\right) \overbrace{\nabla_{\theta_{\mathrm{s},l}} \hat{Y}_\tau(\theta_{\mathrm{s},l})}^{\Psi_\tau(\theta_{\mathrm{s},l}) :=} \\ &= \frac{2}{N_{\mathrm{est}} - \tau} \left(\hat{Y}_\tau(\theta_{\mathrm{s},l})^\top - Y_{\mathrm{m}}(\tau)^\top\right) \Psi_\tau(\theta_{\mathrm{s},l}).\end{aligned} \tag{S38}$$

The computation of Ψ_τ is discussed in the following subsection. The Gauss-Newton iterations for a given time-horizon τ is given by

$$(\hat{\theta}_{\mathrm{s},l})_\tau^{i+1} = (\hat{\theta}_{\mathrm{s},l})_\tau^i + \left(\Psi_\tau((\hat{\theta}_{\mathrm{s},l})_\tau^i)^\top \Psi_\tau((\hat{\theta}_{\mathrm{s},l})_\tau^i)\right)^{-1} \Psi_\tau((\hat{\theta}_{\mathrm{s},l})_\tau^i)^\top \left(Y_{\mathrm{m}}(\tau) - \hat{Y}_\tau((\hat{\theta}_{\mathrm{s},l})_\tau^i)\right) \tag{S39}$$

The Gauss-Newton method is said to have converged when

$$|J_\tau((\hat{\theta}_{\mathrm{s},l})_\tau^i) - J_\tau((\hat{\theta}_{\mathrm{s},l})_\tau^{i-1})| \leq \epsilon_{\mathrm{s}}, \tag{S40}$$

where ϵ_{s} is a user-specified tolerance threshold. Alternatively, the Gauss-Newton algorithm terminates after reaching a maximum number of iterations (to be specified by the user). The converged parameter estimate for time horizon τ is denoted by $(\hat{\theta}_{\mathrm{s},l})_\tau^*$. After convergence of the inner loop, set $\tau = \tau + 1$ and initialize $(\hat{\theta}_{\mathrm{s},l})_\tau^0 = (\hat{\theta}_{\mathrm{s},l})_{\tau-1}^*$, and repeat the Gauss-Newton method. The outer loop is terminated when the following criterion is satisfied

$$|J_\tau((\hat{\theta}_{\mathrm{s},l})_\tau^*) - J_\tau((\hat{\theta}_{\mathrm{s},l})_{\tau-1}^*)| \leq \epsilon_{\mathrm{s}}, \tag{S41}$$

or when the outer loop exceeds the maximum number of iterations. The final parameter estimate is denoted as ${\theta_{\mathrm{s},l}}^*$.

3) Computation of the MSP output and its Jacobian: Computation of the multi-step-ahead prediction model in (S32) for large time-horizons may be infeasible since the model expression may grow polynomially in length and degree due to the numerous recursive substitutions. In order to circumvent this problem, the authors in [10] proposed an iterative scheme to compute the multi-step-ahead prediction output $\hat{Y}_\tau(\theta_{\mathrm{s},l})$ and its Jacobian $\Psi_\tau(\theta_{\mathrm{s},l})$. In this approach, for a given parameter vector $\hat{\theta}_{\mathrm{s},l}$, the τ-step-ahead prediction output $\hat{Y}_\tau(\hat{\theta}_{\mathrm{s},l})$ and Jacobian $\Psi_\tau(\hat{\theta}_{\mathrm{s},l})$ are computed recursively based on $\hat{Y}_l(\hat{\theta}_{\mathrm{s},l})$, $\hat{Y}_{l+1}(\hat{\theta}_{\mathrm{s},l}), \ldots, \hat{Y}_{\tau-1}(\hat{\theta}_{\mathrm{s},l})$.

For a given parameter vector $\hat{\theta}_{\mathrm{s},l}$, the l-step-ahead predicted output is given by the linear-in-the-parameters formulation

$$\hat{y}(k|k-l) = \varphi_l(k)^\top \hat{\theta}_{\mathrm{s},l}. \tag{S42}$$

where $\varphi_l(k), (k > l)$ are the rows of matrix ϕ_l and contain monomials in u_{m} and y_{m}. Hence, $\hat{Y}_l(\hat{\theta}_{\mathrm{s},l})$ can be computed as a matrix inner-product. Based on the $\hat{Y}_l(\hat{\theta}_{\mathrm{s},l})$, the $(l+1)$-step-ahead prediction output can be expressed as

$$\hat{y}(k|k-l-1) = \varphi_{l+1}(k, \hat{\theta}_{\mathrm{s},l})^\top \hat{\theta}_{\mathrm{s},l}, \tag{S43}$$

where $\varphi_{l+1}(k, \hat{\theta}_{\mathrm{s},l}), (k > l+1)$ contains monomials in $u_{\mathrm{m}}(k)$, $\ldots$, $u_{\mathrm{m}}(k - n_u - l - 1)$, $y_{\mathrm{m}}(k-l-1), \ldots, y_{\mathrm{m}}(k-l-n_y)$, and $\hat{y}(k-l|k-l-1)$, and can be computed from $\varphi_l(k)$ using the following substitution

$$\varphi_{l+1}(k, \hat{\theta}_{\mathrm{s},l}) = \varphi_l(k)|_{\forall j < (l+1): y_{\mathrm{m}}(k-j) = \hat{y}(k-j|k-l-1)}. \tag{S44}$$

Observe that, although the $(l+1)$-step-ahead prediction $\hat{y}(k|k-l-1)$ depends non-linearly on $\hat{\theta}_{\mathrm{s},l}$, the non-linear dependency is hidden in the regressor $\varphi_{l+1}(k)$, which can be computed if $\varphi_l(k)$ is known. Hence, using $\hat{Y}_l(\hat{\theta}_{\mathrm{s},l})$, predicted outputs $\hat{Y}_{l+1}(\hat{\theta}_{\mathrm{s},l})$ can be computed as a matrix inner-product.

This concept can be generalized to compute the τ-step-ahead prediction output $\hat{y}(k|k-\tau)$ iteratively, through the following sequence of matrix inner-products

$$\begin{aligned}\hat{y}(k|k-l) &= \varphi_l(k)^\top \hat{\theta}_{\mathrm{s},l}, \\ \hat{y}(k|k-l-1) &= \varphi_{l+1}(k, \hat{\theta}_{\mathrm{s},l})^\top \hat{\theta}_{\mathrm{s},l}, \\ &\vdots \\ \hat{y}(k|k-\tau) &= \varphi_\tau(k, \hat{\theta}_{\mathrm{s},l})^\top \hat{\theta}_{\mathrm{s},l},\end{aligned} \tag{S45}$$

where $\varphi_\tau(k, \hat{\theta}_{\mathrm{s},l})$ is derived from $\varphi_l(k), \ldots, \varphi_l(k, \hat{\theta}_{\mathrm{s},l})$ by performing the following substitution

$$\varphi_\tau(k) = \varphi_l(k)|_{\forall j < \tau: y_{\mathrm{m}}(k-j) = \hat{y}(k-j|k-\tau)}. \tag{S46}$$

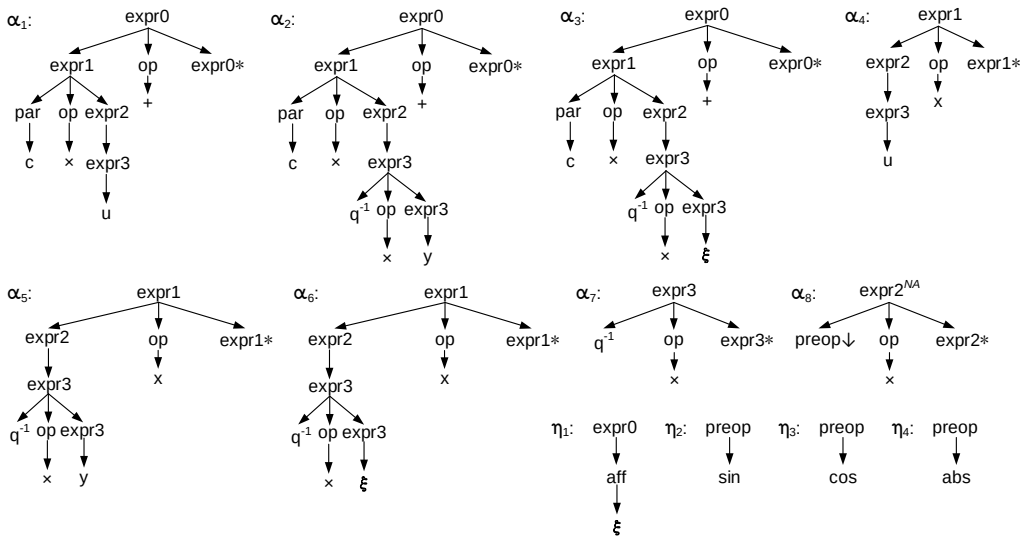


Fig. S5: Initial trees $\{\eta_i\}_{i=1}^{4}$ and auxiliary trees $\{\alpha_j\}_{j=1}^{8}$ of the of the over-arching TAG G_{AT} [5].

The variable substitution in (S46) replaces measured output terms $y_{\mathrm{m}}(k-j)$ in $\varphi_l(k)$ that lie within the prediction horizon $k-(\tau+1),\ldots,k$ with predicted output terms $\hat{y}(k-j|k-\tau)$. This implies that the vector $\varphi_\tau(k)$ consists of monomials in $u_{\mathrm{m}}(k),\ldots,u_{\mathrm{m}}(k-n_u-l)$, $y_{\mathrm{m}}(k-\tau),\ldots,y_{\mathrm{m}}(k-l-n_y)$, and $\hat{y}(k-l-1|k-\tau),\ldots,\hat{y}(k-\tau+1|k-\tau)$. In this manner, performing numerous recursive symbolic substitutions in the non-linear model to compute (S32) can be avoided.

For a given $\hat{\theta}_{\mathrm{s},l}$, the Jacobian of the predicted outputs can also be computed iteratively, as follows. Let $\Psi_{\tau,k}$ denote the Jacobian of the predicted output $\hat{y}(k|k-\tau)$, i.e.,

$$\Psi_{\tau,k} = \nabla_{\theta_{\mathrm{s},l}} \hat{y}(k|k-\tau). \tag{S47}$$

Since the derivatives of measured inputs u_{m} and outputs y_{m} with respect to $\theta_{\mathrm{s},l}$ is 0, we can compute the Jacobian of the l-step-ahead predicted as

$$\Psi_{l,k} = \varphi_l(k)^\top. \tag{S48}$$

For $\tau > l$, we get

$$\begin{aligned}
\Psi_{l+1,k} &= \varphi_{l+1}(k,\hat{\theta}_{\mathrm{s},l})^\top + \hat{\theta}_{\mathrm{s},l}^\top \nabla_{\theta_{\mathrm{s},l}} \varphi_{l+1}(k,\theta_{\mathrm{s},l})\big|_{\theta_{\mathrm{s},l}=\hat{\theta}_{\mathrm{s},l}},\\
&\vdots\\
\Psi_{\tau,k} &= \varphi_{\tau}(k,\hat{\theta}_{\mathrm{s},l})^\top + \hat{\theta}_{\mathrm{s},l}^\top \nabla_{\theta_{\mathrm{s},l}} \varphi_{\tau}(k,\theta_{\mathrm{s},l})\big|_{\theta_{\mathrm{s},l}=\hat{\theta}_{\mathrm{s},l}}.
\end{aligned} \tag{S49}$$

Recall that $\varphi_{\tau,\hat{\theta}_{\mathrm{s},l}}(k)$ consists of monomials in $u_{\mathrm{m}}(k),\ldots,u_{\mathrm{m}}(k-n_u-l)$, $y_{\mathrm{m}}(k-\tau),\ldots,y_{\mathrm{m}}(k-l-n_y)$, and $\hat{y}(k-l-1|k-\tau),\ldots,\hat{y}(k-\tau+1|k-\tau)$. Hence, $\Psi_{\tau,k}$ can be computed iteratively since for all $j\in[l+1,\tau-1]$, we have $\nabla_{\theta_{\mathrm{s},l}}\hat{y}(k-j|k-\tau) = \Psi_{\tau-j,k-j}$.

S6. Simulation Results

The initial and auxiliary trees of TAG G_{AT} used in the paper are illustrated in Fig. S5. The proposed TAG extends beyond the class of P-NARMAX by including $\sin$, $\cos$ and abs functions of the input, output and noise terms. Note that the trigonometric and absolute-value non-linearities are introduced *without* scaling of their arguments, thereby retaining the linear-in-the-parameters structure of the models.

For the simulation example considered in the paper, we collect the results obtained from the Monte-Carlo (MC) simulations of the proposed method with various rates of crossover and mutation. In Fig. S6 – S10, we illustrate the results obtained for the different values of mutation rate p_m. In Fig. S11 – S14, we illustrate the results obtained for the different values of crossover rate p_m. For a discussion on these Figures, see Sec. 7.3 of the paper.

References

[1] D. Khandelwal, M. Schoukens, and R. Tóth, "A tree adjoining grammar representation for models of stochastic dynamical systems," 2020.

[2] N. X. Hoai and R. McKay, "A framework for tree adjunct grammar guided genetic programming," in *Proceedings of the Post-graduate ADFA Conference on Computer Science (PACCS'01)*, 2001, pp. 93–99.

[3] J. R. Koza, *Genetic programming: on the programming of computers by means of natural selection*. MIT press, 1992, vol. 1.

[4] D. Ashlock, C. McGuinness, and W. Ashlock, "Representation in evolutionary computation," in *IEEE World Congress on Computational Intelligence*. Springer, 2012, pp. 77–97.

[5] K. Deb, A. Pratap, S. Agarwal, and T. Meyarivan, "A fast and elitist multiobjective genetic algorithm: Nsga-ii," *IEEE transactions on evolutionary computation*, vol. 6, no. 2, pp. 182–197, 2002.

[6] S. A. Billings, *Nonlinear system identification: NARMAX methods in the time, frequency, and spatio-temporal domains*. John Wiley & Sons, 2013.

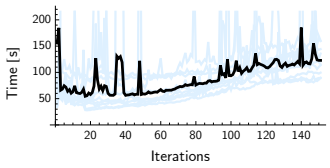
[7] S. Billings and W. Voon, "Least squares parameter estimation algorithms for non-linear systems," *International Journal of Systems Science*, vol. 15, no. 6, pp. 601–615, 1984.

[8] D. Khandelwal, M. Schoukens, and R. Tóth, "On the simulation of polynomial narmax models," in *2018 IEEE Conference on Decision and Control (CDC)*. IEEE, 2018, pp. 1445–1450.

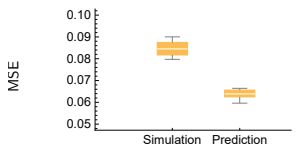
[9] M. Farina and L. Piroddi, "Simulation error minimization identification based on multi-stage prediction," *International Journal of Adaptive Control and Signal Processing*, vol. 25, no. 5, pp. 389–406, 2011.

[10] ——, "Identification of polynomial input/output recursive models with simulation error minimisation methods," *International Journal of Systems Science*, vol. 43, no. 2, pp. 319–333, 2012.

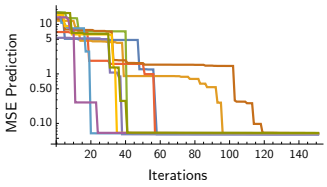
[5] The superscript *NA* in auxiliary tree α_8 refers to a *null adjunction* constraint on the root node, which prohibits the adjunction of any auxiliary tree at that location. See [11] for details.



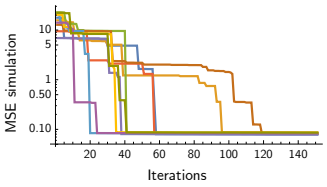
(a) Average time taken per generation for each GP run (light blue) and mean over the 10 runs (black).



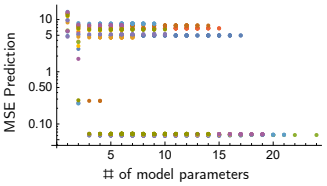
(b) Distribution of the minimum mean squared error (MSE) prediction and simulation error evaluated on $\mathcal{D}_{\text{test}}$ over the 10 runs.



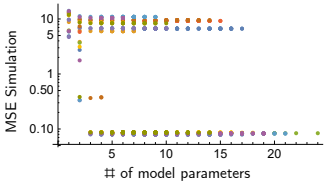
(c) Evolution of minimum prediction error w.r.t GP iterations.



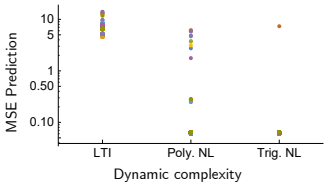
(d) Evolution of minimum simulation error w.r.t GP iterations.



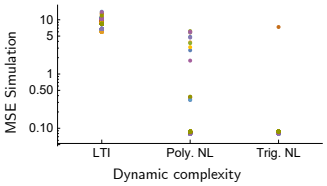
(e) Pareto front - prediction error vs parametric complexity.



(f) Pareto front - simulation error vs parametric complexity.

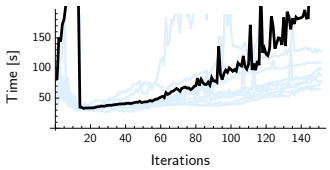


(g) Pareto front - prediction error vs dynamic complexity.

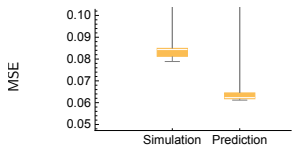


(h) Pareto front - simulation error vs dynamic complexity.

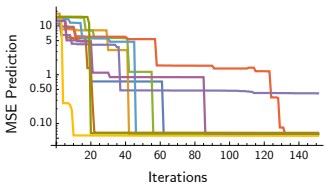
Fig. S6: MC simulation results for the academic example with $p_m = 1$ and $p_c = 0.8$.



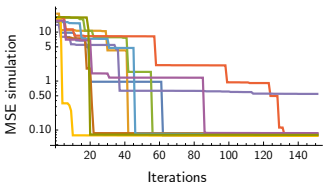
(a) Average time taken per generation for each GP run (light blue) and mean over the 10 runs (black).



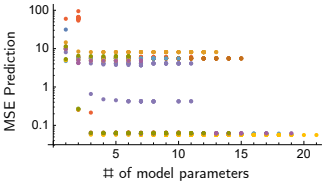
(b) Distribution of the minimum MSE prediction and simulation error evaluated on $\mathcal{D}_{\text{test}}$ over the 10 runs (1 outlier not in image).



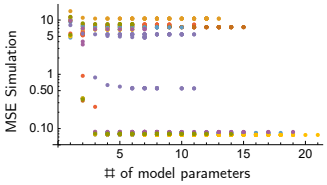
(c) Evolution of minimum prediction error w.r.t GP iterations.



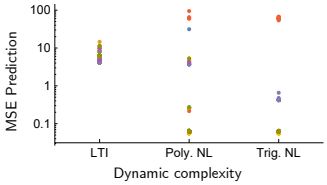
(d) Evolution of minimum simulation error w.r.t GP iterations.



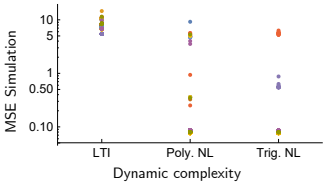
(e) Pareto front - prediction error vs parametric complexity.



(f) Pareto front - simulation error vs parametric complexity.

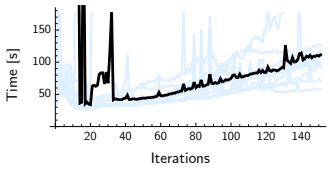


(g) Pareto front - prediction error vs dynamic complexity.

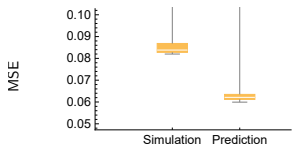


(h) Pareto front - simulation error vs dynamic complexity.

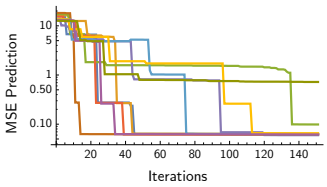
Fig. S7: MC simulation results for the academic example with $p_m = 0.6$ and $p_c = 0.8$.



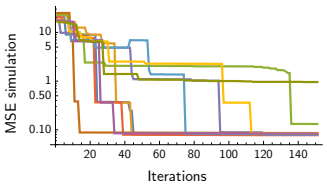
(a) Average time taken per generation for each GP run (light blue) and mean over the 10 runs (black).



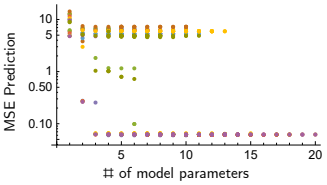
(b) Distribution of the minimum MSE prediction and simulation error evaluated on $\mathcal{D}_{\text{test}}$ over the 10 runs (2 outliers not in image).



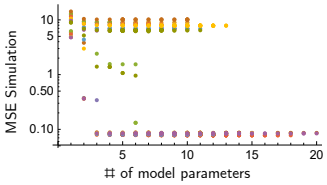
(c) Evolution of minimum prediction error w.r.t GP iterations.



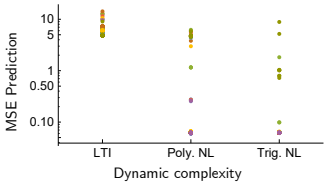
(d) Evolution of minimum simulation error w.r.t GP iterations.



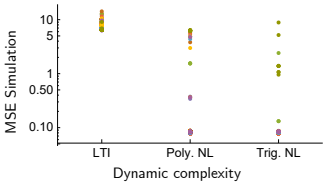
(e) Pareto front - prediction error vs parametric complexity.



(f) Pareto front - simulation error vs parametric complexity.

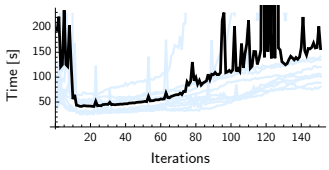


(g) Pareto front - prediction error vs dynamic complexity.

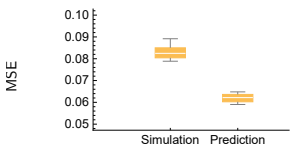


(h) Pareto front - simulation error vs dynamic complexity.

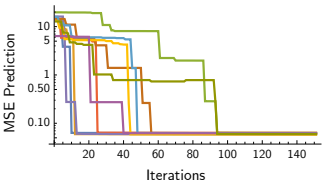
Fig. S8: MC simulation results for the academic example with $p_m = 0.4$ and $p_c = 0.8$.



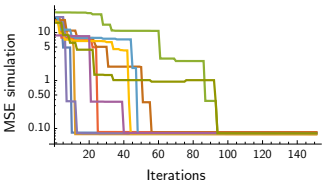
(a) Average time taken per generation for each GP run (light blue) and mean over the 10 runs (black).



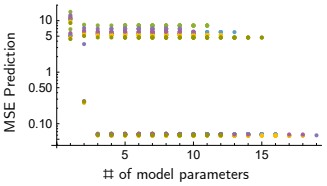
(b) Distribution of the minimum MSE prediction and simulation error evaluated on $\mathcal{D}_{\text{test}}$ over the 10 runs.



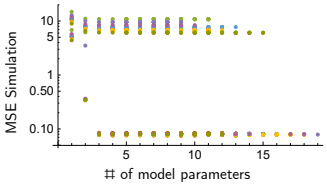
(c) Evolution of minimum prediction error w.r.t GP iterations.



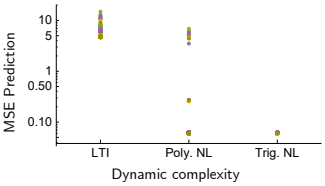
(d) Evolution of minimum simulation error w.r.t GP iterations.



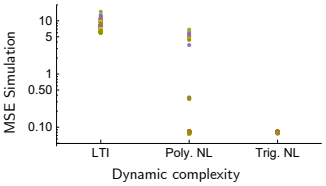
(e) Pareto front - prediction error vs parametric complexity.



(f) Pareto front - simulation error vs parametric complexity.

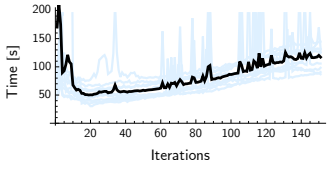


(g) Pareto front - prediction error vs dynamic complexity.

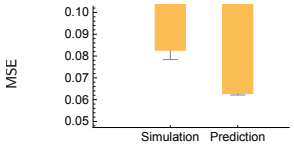


(h) Pareto front - simulation error vs dynamic complexity.

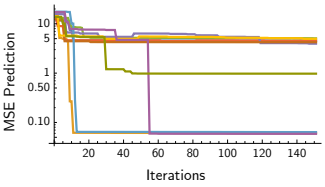
Fig. S9: MC simulation results for the academic example with $p_m = 0.2$ and $p_c = 0.8$.



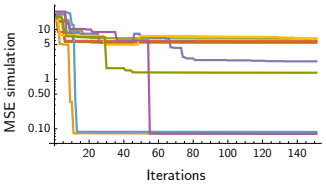
(a) Average time taken per generation for each GP run (light blue) and mean over the 10 runs (black).



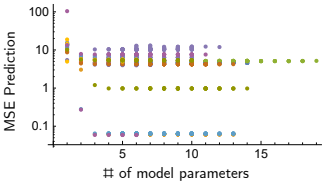
(b) Distribution of the minimum MSE prediction and simulation error evaluated on $\mathcal{D}_{\text{test}}$ over the 10 runs (7 instances not in image).



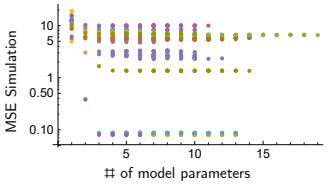
(c) Evolution of minimum prediction error w.r.t GP iterations.



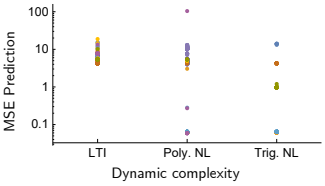
(d) Evolution of minimum simulation error w.r.t GP iterations.



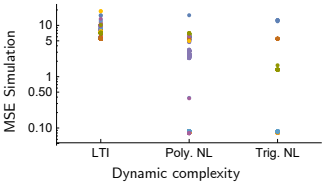
(e) Pareto front - prediction error vs parametric complexity.



(f) Pareto front - simulation error vs parametric complexity.

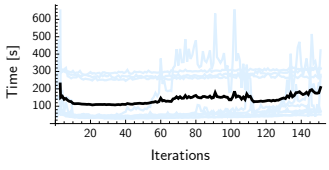


(g) Pareto front - prediction error vs dynamic complexity.

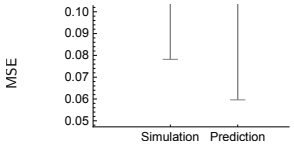


(h) Pareto front - simulation error vs dynamic complexity.

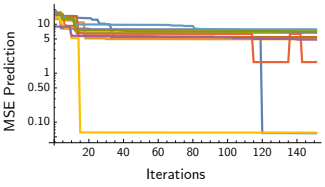
Fig. S10: MC simulation results for the academic example with $p_m = 0$ and $p_c = 0.8$.



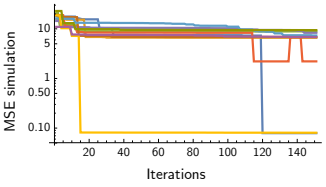
(a) Average time taken per generation for each GP run (light blue) and mean over the 10 runs (black).



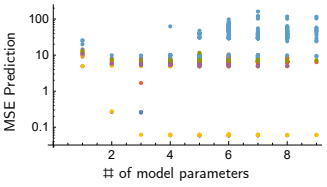
(b) Distribution of the minimum MSE prediction and simulation error evaluated on $\mathcal{D}_{\text{test}}$ over the 10 runs (8 instances not in image).



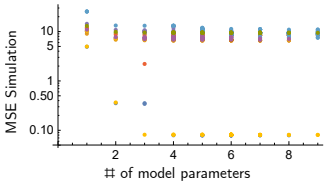
(c) Evolution of minimum prediction error w.r.t GP iterations.



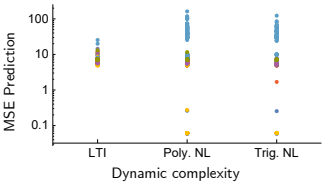
(d) Evolution of minimum simulation error w.r.t GP iterations.



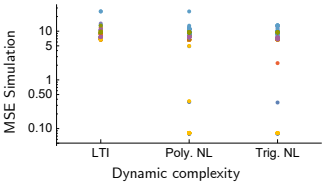
(e) Pareto front - prediction error vs parametric complexity.



(f) Pareto front - simulation error vs parametric complexity.

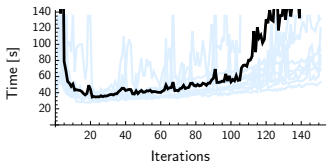


(g) Pareto front - prediction error vs dynamic complexity.

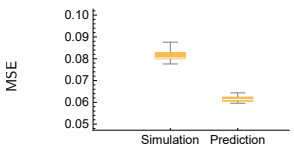


(h) Pareto front - simulation error vs dynamic complexity.

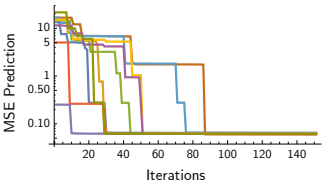
Fig. S11: MC simulation results for the academic example with $p_m = 0.8$ and $p_c = 1$.



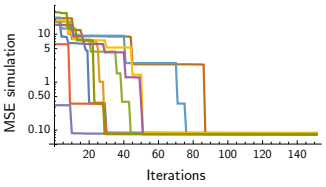
(a) Average time taken per generation for each GP run (light blue) and mean over the 10 runs (black).



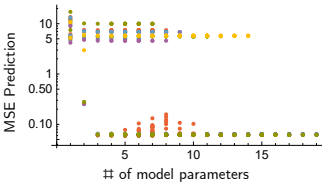
(b) Distribution of the minimum MSE prediction and simulation error evaluated on $\mathcal{D}_{\text{test}}$ over the 10 runs.



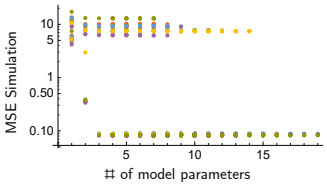
(c) Evolution of minimum prediction error w.r.t GP iterations.



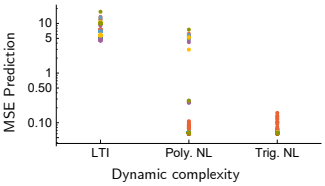
(d) Evolution of minimum simulation error w.r.t GP iterations.



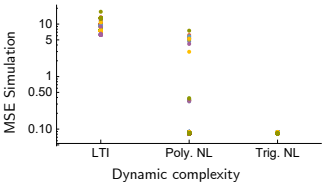
(e) Pareto front - prediction error vs parametric complexity.



(f) Pareto front - simulation error vs parametric complexity.

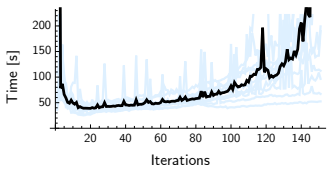


(g) Pareto front - prediction error vs dynamic complexity.

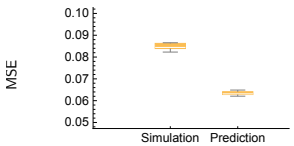


(h) Pareto front - simulation error vs dynamic complexity.

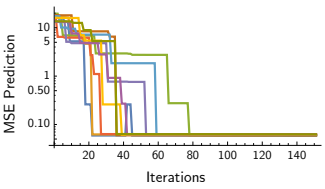
Fig. S12: MC simulation results for the academic example with $p_m = 0.8$ and $p_c = 0.6$.



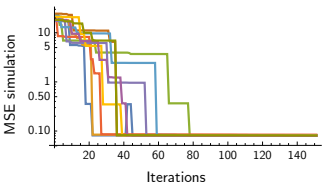
(a) Average time taken per generation for each GP run (light blue) and mean over the 10 runs (black).



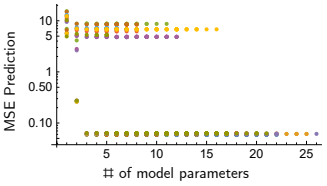
(b) Distribution of the minimum MSE prediction and simulation error evaluated on $\mathcal{D}_{\text{test}}$ over the 10 runs.



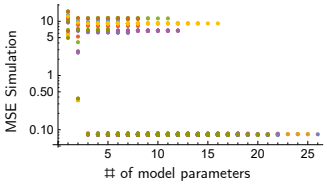
(c) Evolution of minimum prediction error w.r.t GP iterations.



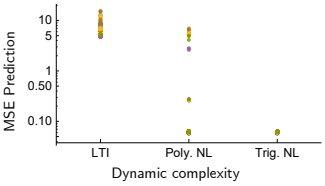
(d) Evolution of minimum simulation error w.r.t GP iterations.



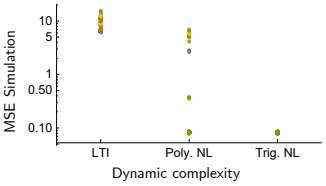
(e) Pareto front - prediction error vs parametric complexity.



(f) Pareto front - simulation error vs parametric complexity.

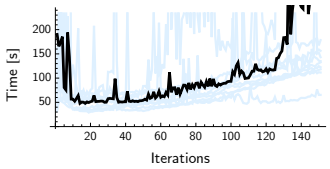


(g) Pareto front - prediction error vs dynamic complexity.

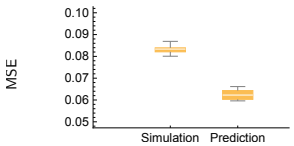


(h) Pareto front - simulation error vs dynamic complexity.

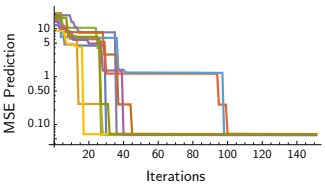
Fig. S13: MC simulation results for the academic example with $p_m = 0.8$ and $p_c = 0.4$.



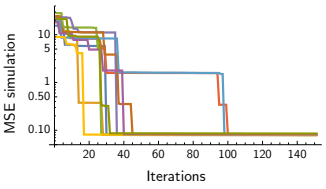
(a) Average time taken per generation for each GP run (light blue) and mean over the 10 runs (black).



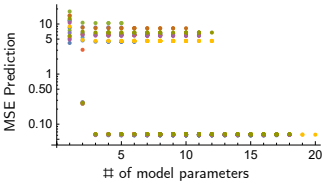
(b) Distribution of the minimum MSE prediction and simulation error evaluated on $\mathcal{D}_{\text{test}}$ over the 10 runs.



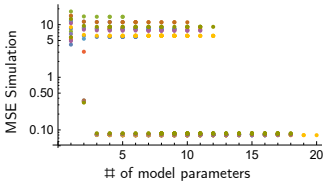
(c) Evolution of minimum prediction error w.r.t GP iterations.



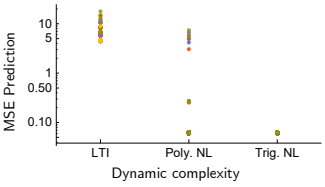
(d) Evolution of minimum simulation error w.r.t GP iterations.



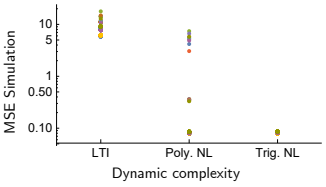
(e) Pareto front - prediction error vs parametric complexity.



(f) Pareto front - simulation error vs parametric complexity.

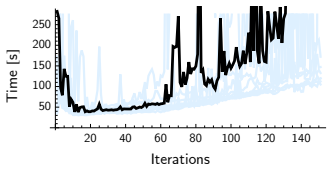


(g) Pareto front - prediction error vs dynamic complexity.

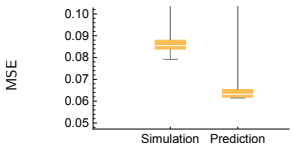


(h) Pareto front - simulation error vs dynamic complexity.

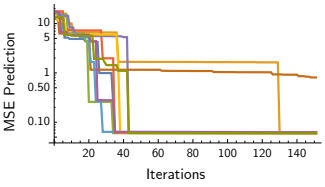
Fig. S14: MC simulation results for the academic example with $p_m = 0.8$ and $p_c = 0.2$.



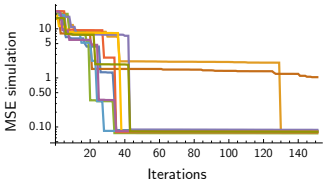
(a) Average time taken per generation for each GP run (light blue) and mean over the 10 runs (black).



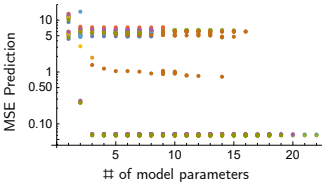
(b) Distribution of the minimum MSE prediction and simulation error evaluated on $\mathcal{D}_{\text{test}}$ over the 10 runs (1 outlier not in image).



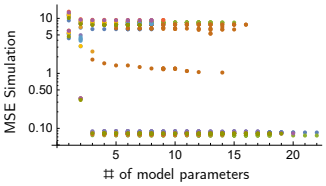
(c) Evolution of minimum prediction error w.r.t GP iterations.



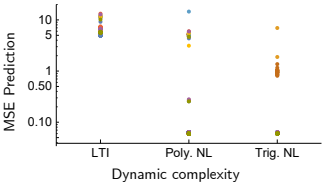
(d) Evolution of minimum simulation error w.r.t GP iterations.



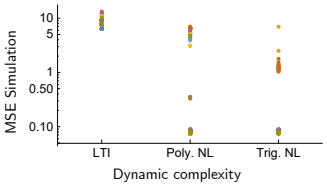
(e) Pareto front - prediction error vs parametric complexity.



(f) Pareto front - simulation error vs parametric complexity.



(g) Pareto front - prediction error vs dynamic complexity.



(h) Pareto front - simulation error vs dynamic complexity.

Fig. S15: MC simulation results for the academic example with $p_m = 0.8$ and $p_c = 0$.

[11] A. K. Joshi and Y. Schabes, "Tree-adjoining grammars," in *Handbook of formal languages*. Springer, 1997, pp. 69–123.