Computers and Mathematics with Applications 64 (2012) 3763–3776

Contents lists available at SciVerse ScienceDirect

Computers and Mathematics with Applications

journal homepage: www.elsevier.com/locate/camwa

An agent-based model of hierarchic genetic search

Robert Schaefer [a], Aleksander Byrski [a,*], Joanna Kołodziej [b], Maciej Smołka [c]

[a] AGH University of Science and Technology, Al. Mickiewicza 30, 30-059 Kraków, Poland

[b] University of Bielsko-Biala, ul. Willowa 2, 43-309 Bielsko-Biała, Poland

[c] Jagiellonian University, ul. Łojasiewicza 6, 30-348 Kraków, Poland

ARTICLE INFO

Keywords:
Genetic algorithms
Multi-agent genetic system
Global optimisation
Hierarchic genetic strategy

ABSTRACT

An effective exploration of the large search space by single population genetic-based metaheuristics may be a very time consuming and complex process, especially in the case of dynamic changes in the system states. Speeding up the search process by the metaheuristic parallelisation must have a significant negative impact on the search accuracy.

There is still a lack of complete formal models for parallel genetic and evolutionary techniques, which might support the parameter setting and improve the whole (often very complex) structure management.

In this paper, we define a mathematical model of Hierarchical Genetic Search (HGS) based on the genetic multi-agent system paradigm. The model has a decentralised population management mechanism and the relationship among the parallel genetic processes has a multi-level tree structure. Each process in this tree is Markov-type and the conditions of the commutation of the Markovian kernels in HGS branches are formulated.

1. Introduction

An effective exploration of the large search space by single population *Genetic Algorithms (GAs)* may be a very time consuming and complex process, especially in the case of dynamic changes in the system state (grid, cloud and network computing) and dynamic boundary conditions of the considered optimisation problem. Speeding up the search process must have a significant negative impact on the search accuracy. The parallelisation of standard GAs is the simplest solution for the improvement of the genetic search effectiveness, especially in heterogeneous environments. Thus complex systems may be created, e.g. leveraging concepts of agency and evolution, applied to various problems, standard ones like global optimisation [1,2], or sophisticated ones, like intrusion detection [3].

The island GA model [4] is the most popular parallel GA approach in global optimisation. However, the effectiveness of the island GA search is very low in the regions with the 'low-quality' local optima in the sense of global optimisation of the considered objective(s). In such cases *Hierarchic Genetic Strategy (HGS)* with its tree structure has proven its effectiveness in solving very complex continuous, discrete and combinatorial optimisation problems in static and dynamic environments [5–7].

The HGS method has been introduced by Schaefer and Kołodziej [5] as a multipopulation genetic strategy with an adaptive (various) search accuracy at the various HGS tree levels. This can be achieved by increasing the density of the binary or affine coding meshes and reducing the mutation parameter values (mutation probabilities or standard deviation of the mutation probability distribution) in the HGS branches.

A concept of the genetic search with an adaptive modular accuracy has been intensively studied in the literature. The adaptive search mechanism in HGS is similar to *Dynamic Parameter Encoding (DPE)* method proposed by Schraudolph and

* Corresponding author. Tel.: +48 126174452.
E-mail addresses: schaefer@agh.edu.pl (R. Schaefer), olekb@agh.edu.pl (A. Byrski), jkolodziej@ath.bielsko.pl (J. Kołodziej), smolka@ii.uj.edu.pl (M. Smołka).

0898-1221/$ – see front matter © 2012 Elsevier Ltd. All rights reserved.
doi:10.1016/j.camwa.2012.02.052

Belew in [8], where the accuracy of encoded parameters is dynamically adjusted to increase the resolution of the solution and to focus the search in the most promising region of the search space. The search outside the chosen region is then suspended.

Another example is *Delta Coding* algorithm introduced by Whitley et al. [9]. In this method after each run of single population genetic algorithm (GENITOR algorithm is applied) the diversity of the population is checked and monitored by measuring of the Hamming distance between each of the parameter in the best and worst strings in population. Then a delta iteration is initialised, where the length of codes of individuals in the new population is reduced (typically by single bit for each parameter). Whitley applied Delta coding strategy in parallel in the island model.

The search accuracy modulated by the dynamic mutation and crossover parameters was proposed as a main genetic mechanism in *Adaptive Genetic Algorithms, (AGAs)* [10]. The probabilities of crossover and mutation greatly determine the degree of solution accuracy, because AGAs utilise the population information in each generation and adaptively adjust the genetic operators' parameters in order to maintain the population diversity as well as to sustain the convergence capacity. This adjustment of crossover and mutation probabilities depends on the fitness values of the solutions. This procedure is different in *Clustering-based Adaptive Genetic Algorithm (CAGA)* [11], where a simple cluster analysis is used to judge the optimisation states of the population, and then the adjustment of genetic operators' parameters depends on these optimisation states. In many recent works to improve the search adaptation mechanism the GA's framework is combined with the *Artificial Neural Network (ANN)* mechanism and fuzzy clustering technologies [12].

The search process in HGS starts by activating a mono-population GA with the low accuracy of search, which governs the whole search process and is responsible for the detection of promising partial solutions and exploration of the new unrecognised regions in an optimisation domain. This process is called a *root* of the HGS tree structure. The more accurate processes are activated in the neighbourhoods of the partial solutions found by the core in order to prevent the premature convergence of the strategy and for a possible improvement of the best found solutions. The activation of new processes does not increase significantly the complexity of the hierarchic strategy because of three main reasons: (i) differently to the hybrid strategies, where the components are usually composed of various metaheuristics and local search methods, we use the same general framework for the algorithms working at all levels of the HGS; (ii) the HGS tree extension is steered by the specialised operations responsible for the deactivation of the ineffective processes and by the effectiveness of the search in the core of the tree; (iii) Finally, the synchronisation of the search is provided with use of predefined agents acting at the nodes of the HGS tree, that in the advanced cases may be also responsible for e.g., load balancing.

In this paper we develop a formal agent-based mathematical model for HGS. Our contributions include:

- A development of a formal agent-based model for HGS strategy.
- A design of the agent actions and mechanisms associated with the main HGS branching and genetic operators.

The rest of the paper is organised as follows. In Section 2 we define a basic formal model for single population GAs, which is then used in modelling the whole strategy. The HGS basic operators parameters and the main concept of its agent-based model are presented in Sections 4.1, 4.2, 5, 6, 6.1 and 7. We conclude the paper in Section 8.

2. Preliminaries

2.1. Global optimisation problem

A general global optimisation problem addressed in this paper can be defined as follows.

Definition 2.1. Let us denote by $\mathcal{D} \subset \mathbb{R}^N$ the admissible domain and by $\xi \in \mathcal{D}$ the vector decision variable. We assume that $\mathcal{D}$ is compact and regular in the space $(\mathbb{R}^N; \|\cdot\|)$, where $\|\cdot\|$ is the standard Euclidean measure. An *objective* is a bounded, real-valued function $\Phi : \mathcal{D} \to \mathbb{R}$ such that

$$\forall \xi \in \mathcal{D}\ c \leq \Phi(\xi) \leq C \tag{1}$$

for some fixed $c, C \in \mathbb{R}$. A *global minimisation problem* is defined as the problem of finding all $\hat{\xi} \in \mathcal{D}$ satisfying:

$$\hat{\xi} = \arg\min_{\xi \in \mathcal{D}} \{\Phi(\xi)\}. \tag{2}$$

It can be observed that the problem of the minimisation of the objective Φ can be easily transformed into the maximisation of $\tilde{\Phi}(\xi) = -\Phi(\xi) + (C - c);\ \forall \xi \in \mathcal{D}$.

2.2. Population and individuals

A simple single-population GA is usually specified as a process of transformation of *populations* of individuals. Each population P is defined as a finite multi-set of elements selected (with repetitions) from U. It means that the set P may contain many clones of a given genotype (few genotypes may have the same genetic representation). Formally, the population P is defined as a tuple $P = (U, \eta_P)$, where $\eta_P : U \to \mathbb{N} \cup \{0\}$ denotes a function, which counts the number of the clones of genotypes in P. The cardinality $\mu = \#P$ of population P is computed as the sum of the values of the function η_P for all different genotypes appearing in P, i.e. $\mu = \sum_{\xi \in U} \eta_P(\xi)$ (see, e.g., [13]). If U and P are finite sets, then the population is

usually represented by its *frequency vector* x of appearance of all possible genotypes from U in P, that is to say:

$$x = \frac{1}{\mu}\left[\eta_P(\xi_1), \ldots, \eta_P(\xi_r)\right]^T, \tag{3}$$

where $\mathbb{N} \ni r = \#U$, and $U = \{\xi_1, \ldots, \xi_r\}$. The coordinates of vector x are identical with the barycentric coordinates of elements of the standard unit $(r-1)$-dimensional simplex Λ^r associated with the finite genetic universum U, which is defined as follows:

$$\mathbb{R}^r \supset \Lambda^r = \left\{ x \in \mathbb{R}^r : 0 \le x_\xi \le 1,\ \forall\, \xi \in U,\ \sum_{\xi \in U} x_\xi = 1 \right\}. \tag{4}$$

Note that Λ^r contains all frequency vectors of all possible populations of individuals from U.

Let us denote by X_μ^r the finite subset of points in Λ^r corresponding to all populations of the size μ. The set X_μ^r becomes dense in Λ^r when $\mu \to \infty$ (see [14]). Notice, that Λ^r can be also identified with the space of all probabilistic measures $\mathcal{M}(U)$ on the set of genotypes (see, e.g., [15,13]). Hereafter we will denote by $\mathcal{M}(A)$ the space of probabilistic measures defined on a Σ-algebra over the set A. To avoid confusion we formally introduce a one-to-one mapping

$$\Theta : \Lambda^r \ni x \to \Theta(x) \in \mathcal{M}(U) \tag{5}$$

that allows us to identify a frequency population vector $x \in \Lambda^r$ with the related probabilistic measure $\Theta(x) \in \mathcal{M}(U)$.

2.3. Genetic operators and succession schemes

In order to describe comprehensively the dynamics of finite and infinite population genetic algorithms with finite genetic universa, we will consider the *selection operator* $F : \Lambda^r \to \Lambda^r$ that represents the stochastic effect of selection and the *mixing operator* $M : \Lambda^r \to \Lambda^r$ associated with the genetic operations, namely the crossover coupled with the mutation (see [14,16,15]).

The ξ-th coordinate of the selection operator returns the probability of selecting an individual with the genotype $\xi \in U$ from the population represented by the vector $x \in \Lambda^r$. Formally, we denote this probability as $\Theta_\xi(F(x))$. Similarly $\Theta_\xi(M(x))$ is the probability of obtaining individual with the genotype $\xi \in U$ by mixing from the population represented by the vector $x \in \Lambda^r$.

In the case of genetic universum composed of binary strings (e.g. the SGA case, $U = \Omega$ [16]) the selection operators imposed by proportional, tournament and ranking selection schemes are described in [16, Section 4.2], whereas the mixing consisted in binary crossover and positional, bitwise mutation in [16, Sections 4.3–4.5].

Observation 2.1. *Given a probability distribution $\rho \in \mathcal{M}(U)$ such that ρ_ξ is the probability of obtaining the individual with the genotype $\xi \in U$ in the next epoch, the probability of obtaining the next population of cardinality $\mu < +\infty$, represented by the vector $y \in X_\mu^r$ is given by the polynomial probability distribution*

$$\mathrm{Pr}_\rho^\mu(y) = \frac{\mu!}{\prod_{\xi \in U}(\mu \cdot y_\xi)!} \prod_{\xi \in U} (\rho_\xi)^{\mu \cdot y_\xi}. \tag{6}$$

We will apply two succession schemes (the schemes of obtaining the next epoch population from the current one). The first one will be called *Standard Succession Scheme* (SSS) and consists of selecting the intermediate population of parental individuals of the same cardinality μ, by independent μ-fold sampling according to the probability distribution $\Theta(F(x))$. The probability of obtaining the intermediate population might be computed using formula (6) by setting $\rho = \Theta(F(x))$, where $x \in X_\mu^r$ stands for the current population vector. The next-epoch population is then derived from the intermediate one by means of admissible genetic operations (the mixing is the composition of those operations). The probability of the next-epoch population may be evaluated again using (6) by setting $\rho = \Theta(M(z))$, where $z \in X_\mu^r$ denotes now the intermediate population vector. Summing up, according to the Bayes rule (see e.g. [17]) the probability of the next-epoch population obtained by the standard succession rule is described by the following function

$$\tau_{St} : X_\mu \times X_\mu \ni (x, y) \to \tau_{St}(x, y) = \sum_{z \in X_\mu} \mathrm{Pr}^\mu_{\Theta(F(x))}(z) \cdot \Gamma_{zy}, \tag{7}$$

where the $n \times n$ matrix Γ is given by the formula:

$$\Gamma_{xy} = \mathrm{Pr}^\mu_{\Theta(M(x))}(y), \quad x, y \in X_\mu^r. \tag{8}$$

In *Vose Succession Scheme* (VSS), there is a heuristic operator: $G = M \circ F : \Lambda^r \to \Lambda^r$ introduced by [16] for the Simple Genetic Algorithm. The probability of the next epoch population is given now by the following function

$$\tau_G : X_\mu^r \times X_\mu^r \ni (x, y) \to \tau_G(x, y) = \mathrm{Pr}^\mu_{\Theta(G(x))}(y). \tag{9}$$

The VSS might be implemented by μ-fold execution of the following three steps [16, Chapter 5]:

1. select two parental individuals from the current population,

2. mutate each of them,
3. cross the mutated parents and add one randomly selected child to the next epoch population.

Because X_μ^r is finite ($\#X_\mu^r = n < +\infty$), the transition probability functions (7), (9) can be represented by a transition matrix Q in the following way: for each pair of population vectors $x, y \in X_\mu^r$ we have

$$Q_{xy} = \begin{cases} \tau_{St}(x, y) & \text{if SSS is applied,} \\ \tau_G(x, y) & \text{if VSS is used.} \end{cases} \tag{10}$$

The interpretation of the matrix Q depends on the context in which the succession scheme is determined.

Observation 2.2. *The transition probability function in kstep $\in \mathbb{N}$ epochs of evolution $\tau_{kstep} : X_\mu^r \to \mathcal{M}(X_\mu^r)$ is defined by using the following formula:*

$$\tau_{kstep}(x, y) = ((Q)^{kstep})_{xy}, \quad \forall x, y \in X_\mu^r. \tag{11}$$

2.4. The stopping rule based on the search efficiency

To define a fair stopping condition for GAs may be still a challenging task, especially in the case of the limited abilities of the exploration and exploitation of a huge search space. In such a case the algorithm is stopped after the execution of the *kstep* of evolution epochs if there is no significant improvement in the fitness optimisation. It means that for the following population vectors $x^{(t)}$, $x^{(t+kstep)} \in X_\mu^r$, the GA procedure will be continued if the following condition is satisfied:

$$(x^{(t)} - x^{(t+kstep)}, f) \geq \text{lsc} > 0, \tag{12}$$

where $\text{lsc} \in \mathbb{R}_+$ denotes the stopping threshold parameter. GA is stopped, if

$$(x^{(t)} - x^{(t+kstep)}, f) < \text{lsc}. \tag{13}$$

Hereafter $(\cdot, \cdot)$ will denote the Euclidean scalar product in $\mathbb{R}^r$.

Observation 2.3. *Let us assume that $x \in X_\mu^r$ is the current population. We can calculate the probability of stopping GA due to the efficiency condition (see Eq. (12)) by using the following formula:*

$$S(x) = 1 - \sum_{y \in X_\mu^r} \tau_{kstep}(x, y)\,[((x - y, f) - \text{lsc}) \geq 0], \tag{14}$$

where $[\cdot]$ is a "logic expression" operator. Formally, $S(x)$ may be interpreted as the probability of terminating GA after kstep genetic epochs, and lsc *is the stopping criterion parameter.*

Let us denote by A an event when GA passes from the state $x \in X_\mu^r$ to the state $y \in X_\mu^r$ by executing *kstep* genetic epochs, by B an event when the computation is terminated after *kstep* genetic epochs, and by C the event of the continuation of the computation after the execution of *kstep* genetic epochs. Using the Eqs. (11)–(13) we can compute the following conditional probabilities:

$$\begin{aligned} \Pr(B|A) &= \begin{cases} 0 & \text{if} \quad (x - y, f) \geq \text{lsc} \\ \tau_{kstep}(x, y) & \text{if} \quad (x - y, f) < \text{lsc}, \end{cases} \\ \Pr(C|A) &= \begin{cases} 0 & \text{if} \quad (x - y, f) < \text{lsc} \\ \tau_{kstep}(x, y) & \text{if} \quad (x - y, f) \geq \text{lsc}. \end{cases} \end{aligned} \tag{15}$$

Based on Observation 2.3 and using the general formula for conditional probability [17] we can define the following observation:

Observation 2.4. *The probability of stopping the GA process due to the efficiency condition (see Eq. (12)) in the state $y \in X_\mu^r$ is defined as follows:*

$$S(x)\ \tau_{kstep}(x, y)\ [(x - y, f) < \text{lsc}] \tag{16}$$

and the probability of continuation of the GA search process in the state $y \in X_\mu^r$ is defined as follows:

$$(1 - S(x))\ \tau_{kstep}(x, y)\ [(x - y, f) \geq \text{lsc}]. \tag{17}$$

3. Primitives of the HGS model

In the described case we deal with a scalable system that may be easily implemented and run on parallel computation systems, such as clusters or grids. Because of the terminology used in such systems, from now on, the *population* being part of HGS tree, will be called *deme*.

Now, let us introduce the following settings and notions for HGS that will be used later on:

- We define a family of $m \in \mathbb{N}$ genetic universa U_i, $i = 1, \ldots, m$, $\#U_i = r_i \in \mathbb{N}$, $r_i < +\infty$ for all $i = 1, \ldots, m$.
- For $\{U_i\}_{i=1}^m$ we define the following family of associated encoding operators:

$$\mathrm{code}_i : U_i \to \mathcal{D}; \quad i = 1, \ldots, m \tag{18}$$

with the progressive increase of the search accuracy. It means that there exists the following sequence of encoding parameters:

$$\exists\, d_1, \ldots, d_{m-1} \in \mathbb{N}; \qquad r_{i+1} = d_i\, r_i, \quad i = 1, \ldots, m-1 \tag{19}$$

and each d_i estimates the increment rate in the search accuracy in a pair of U_i and U_{i+1} (and thus the encoding resolution or density of the encoding mesh). For binary implementation of HGS (see [5]) we defined a *Nested Coding* procedure, which generates a set of binary encoded meshes associated with binary genetic universa.

- Let us introduce the following sequence of inheritance surjective mappings:

$$\mathrm{inherit}_i : U_i \to U_{i-1}, \quad i = 2, \ldots, m \tag{20}$$

and sets

$$\begin{aligned} U_i|\xi \;&=\; (\mathrm{inherit}_i)^{-1}(\xi) \\ &= \{\zeta \in U_i : \mathrm{inherit}_i(\zeta) = \xi\}, \quad \xi \in U_{i-1},\ i = 2, \ldots, m. \end{aligned} \tag{21}$$

Moreover we assume, that

$$\begin{aligned} &\forall i = 2, \ldots, m,\ \forall \xi \in U_{i-1} \quad \#U_i|\xi = d_{i-1} \\ &\forall i = 2, \ldots, m,\ \forall \xi, \zeta \in U_{i-1} \quad U_i|\xi \cap U_i|\zeta = \emptyset. \end{aligned} \tag{22}$$

By using the $\mathrm{inherit}_i$ each U_i, $i > 1$ can be decomposed into the set of disjoint clusters $U_i|\xi$ indexed by the elements $\xi \in U_{i-1}$ so that $\bigcup_{\xi \in U_{i-1}} U_i|\xi = U_i$. The family of inheritance mapping is strictly associated with the family of the encoding functions.

- We introduce the family of fitness functions

$$f_i : U_i \to [0, \Delta], \quad i = 0, \ldots, m. \tag{23}$$

Each f_i is associated with Φ (e.g. $U_i \ni \xi \to f_i(\xi) = c + \Phi(\mathrm{code}_i(\xi))$) and the following coherency condition holds:

$$\mathrm{code}_i(\xi) = \mathrm{code}_j(\zeta) \Rightarrow f_i(\xi) = f_j(\zeta), \quad \forall i, j \in \{0, \ldots, m\},\ \xi \in U_i, \zeta \in U_j. \tag{24}$$

The value of the constant Δ depends on the values of c and C parameters specified for the considering optimisation problem defined in Eq. (2). Each fitness function f_i may be identified with the vector of its values $f_i = ((f_i)_\xi)_{\xi \in U_i}$.

- We denote by $\mu_1, \ldots, \mu_m$, the sizes of demes and by $X_1 = X_{\mu_1}^{r_1}, \ldots, X_m = X_{\mu_m}^{r_m}$ the demes in HGS branches of degrees $1, \ldots, m$, respectively.
- We denote by $kstep_1, \ldots, kstep_m$ the lengths of "metaepochs" for each type of searches.
- Each GA process in HGS branches is governed by its own selection and mixing operators $F_i, M_i : \Lambda^{r_i} \to \Lambda^{r_i}$ and by one of the possible succession schemes SSS or VSS. Subsequently, the probability transition matrix Q^i can be defined using formulae (7)–(10). Next, analogously as in Observation 2.2 the transition operator $\tau_i : X_i \to \mathcal{M}(X_i)$ such that

$$\tau_i(x, y) = \left((Q^i)^{kstep_i}\right)_{x,y}, \quad i \in \{0, \ldots, m\} \tag{25}$$

can be established. The value $\tau_i(x, y)$ is equal to the probability of passing between the states x and y after $kstep_i$ epochs of evolution in the arbitrary HGS branch of i-th type.

- For each HGS branch but the core we define the efficiency stopping rule parameter lsc_i (see Eqs. (12), (13)) and the function $S_i : X_i \to [0, 1]$ which defines the probability of stopping the genetic process in a given branch of i-th type (see Observation 2.3).

4. The states of HGS Markov model

4.1. HGS tree

The HGS tree constitutes m-level bidirectional graph $HGSTREE = \langle V, E, F\rangle$. The number of child nodes may vary through levels, but on each level it is constant (equals to k_i, $i = 1, \ldots, m-1$). The labelling F assigns simply the path from the root

to each node. Let us define $K_i = \{1, \ldots, k_i\},\ i = 1, \ldots, m-1$ and

$$\begin{aligned} K^1 &= \{1\} \times \prod_{p=1}^{m-1} \{0\}, \qquad K^m = \{1\} \times \prod_{p=1}^{m-1} K_p, \\ K^i &= \{1\} \times \prod_{p=1}^{i-1} K_p \times \prod_{p=i}^{m-1} \{0\}, \quad m > i > 1, \\ K &= \bigcup_{i=1}^{m} K^i, \qquad K^{\text{par}} = K \setminus K^m = \bigcup_{i=1}^{m-1} K^i \end{aligned} \tag{26}$$

where K is the domain of all labels $(F : V \to K)$, K^{par} labels of parental demes and K^m labels of leaf demes respectively. Moreover, we will need the function

$$\begin{aligned} &l : K \to \{1, \ldots, m\} : \\ &l(j) = \begin{cases} m & \text{if} \quad j_i > 0,\ \forall i = 1, \ldots, m \\ 1 \le s < m & \text{if} \quad j_1, \ldots, j_s > 0,\ j_{s+1}, \ldots, j_m = 0 \end{cases} \end{aligned} \tag{27}$$

which returns the length $l(j)$ of the path $j \in K$. The root is the unique node for which $j = (1, 0, \ldots, 0)$ and $l(j) = 1$. The length of the path $l(j)$ determines also the level of the *HGSTREE* in which the node labelled by j is located. Furthermore, we can assign the set of child indices $\mathcal{I}_j \subset K$ to each parental node indexed by j. Namely for $j \in K$ such that $l(j) < m$ we have

$$\mathcal{I}_j = \left\{ \eta \in K^{l(j)+1} \mid \eta_1 = j_1, \ldots, \eta_{l(j)} = j_{l(j)} \right\}. \tag{28}$$

4.2. The space of states

The state of the whole HGS tree is composed of the states of all active and all potentially active demes. All such demes are contained in nodes V of the *HGSTREE* structure described in Section 4.1.

The state of a deme located in the *HGSTREE* will be determined by its deme vector $x^j \in X_{l(j)}$. Moreover, the state of each deme except the root deme will be characterised by the status label which can take values from the set {*inactive*, *new*, *active*, *stopped*}. The status of the root deme may take only two values {*active*, *stopped*}.

A starting deme vector of the root deme is generated by multiple sampling (sampling with return) from U_1 according to a given probability distribution, and its status is primarily set to *active*.

The lower level demes $x^j :\ l(j) > 1$ might take the status:

- *inactive* if it has not been activated yet by the sprouting operation. The deme vector of each *inactive* deme from the i-th level is considered to have the same, arbitrary value from X_i. This setting is performed only from the formal reasons and does not affect the computation result in any way.
- *new* if it is just sprouted by its parental deme. A deme with this status cannot sprout another deme. The status *new* is changed to *active* or *stopped* after having processed the deme's first metaepoch. Sprouting changes the value of x^j as well (the initial setting is removed). The parental deme has to be *active* in at least one of the previous steps.
- *active* if it was *new* or *active* one metaepoch ago and the efficiency condition was not satisfied.
- *stopped* if the efficiency stopping condition held currently or in the past. The deme marked *stopped* once stays *stopped* up to the end of computation and does not change its deme vector. In case of parental demes of the higher order then leaves the status *stopped* is set also when all their child-demes had been activated (i.e. they have status *active* or *stopped*). Such a situation appears very rarely in the computational practice.

Therefore, the HGS space of states may be considered as a subset of the following space:

$$X = \{\textit{active},\ \textit{stopped}\} \times X_1 \times \prod_{i=2}^{m} \left(\prod_{p=1}^{g_i} (\{\textit{inactive}, \textit{new}, \textit{active}, \textit{stopped}\} \times X_i) \right), \tag{29}$$

where $g_i = \prod_{s=1}^{i-1} k_s$. Note that each HGS state is strictly associated with the *HGSTREE* node indexed by $j \in K$ and it has two components: s^j containing its current status and $x^j \in X_{l(j)}$ being the current deme vector of the deme located at node j.

5. Agent-based synchronisation

Proper synchronisation mechanism that allows the demes to work according to the HGS strategy may be formalised as a multi-agent system. Each node being the member of the set V (the set of nodes in the *HGSTREE* structure) is equipped with the governing agent. The agents will be denoted using the same indexing role as in case of deme vectors and statuses, so $\{ag^j\},\ j \in K$.

Entrusting parts of the system to agents allows for further development of the system, making possible introducing various enhancements on its computational (e.g., individual adaptation of the parameters of the local variation operators) and technical level (e.g., load balancing). The presented way of modelling has already been applied to several agent-based computing systems, c.f. [1,2] related to modelling of EMAS [18–20].

Let us assume, that in the initial state, all demes residing in the whole HGS tree are *inactive* instead of root-deme set to be *active*. The status of a deme is encapsulated in its agent, and may be changed by communicating with the agent, i.e. after the agent receives appropriate message, the status of its deme is affected.

In order to simplify the description of the agents' algorithms, we assume, that the elements of the system state, such as demes, inactive agents' indexes (see (42)), agents' statuses are entrusted to agents, so the agents act directly on them, without need to update them directly. Therefore, in the pseudocodes below, change of the agent's state affects directly the set described by (42).

Pseudocode 5.1: ROOT AGENT'S ALGORITHM ($ag^j,\ j \in K^1$)

```
status ← active
deme ← GEN_ROOT()
stopCondition ← false
newagidx ← mind(deme, j)
send(newagidx, ACTIVATE)
b_receive(newagidx, ACTIVATED)
repeat
    children ← GET_STATUS(j, {active, stopped, new})
    for each a ∈ children do send(a, RUNMETA)
    if status = active
      then METAEPOCH()
    for each a ∈ children do b_receive(a, READY)
    for each a ∈ children do send(a, RUNSPROUT)
    if status = active
      then { seed ← b_1(deme)
             newagidx ← mind(deme, j)
             send(newagidx, seed, ACTIVATE)
             b_receive(newagidx, ACTIVATED)
    for each a ∈ children do b_receive(a, SPROUTED)
    if GET_STATUS(j, {inactive}) = ∅
      then status ← stopped
    stopCondition ← CHECK_GLOBAL_STOP()
until stopCondition
for each a ∈ children do send(a, FINISH)
status ← inactive
```

The following functions will be used in the pseudocodes in the further part of this section. They are explained here without giving detailed algorithm:

- *GEN_ROOT*()—generates randomly the root deme by μ_1-times sampling with return from U_1.
- *GEN_DEME*(*i*, seed)—generates randomly the deme by μ_i-times sampling with return from U_i, where according to the given probability distribution $\sigma_{i-1}^{\text{seed}}$ (see (37)).
- *PARENT*(*agentidx*)—returns the index of parent of the $agentidx \in K \setminus K^1 : parentidx \in K : agentidx \in \mathbb{1}_{parentidx}$ (see (28)).
- *GET_STATUS*(*agentidx*, {$status_1, status_2, \ldots$})—returns the indexes of the children of the *agentidx* with selected *statuses*–$\mathbb{1}_{agentidx}$ (see (28)).
- *METAEPOCH*()—runs the metaepoch for the current deme.
- *CHECK_GLOBAL_STOP*()—checks the global stopping condition.
- *CHECK_STOP_COND*()—checks the local stopping condition for chosen deme.

Moreover, we will use the functions $b_i,\ i \in k, l$ and *mind* introduced in the HGS description, see (38), (27), (43), respectively. Messages used in the agent activities described in the pseudocodes are sent and received using proper communication primitives, that make possible asynchronous blocking *b_receive*(*agidx*, *message*) and non-blocking send(*agidx*, *message*) operations. Such primitives operate using message queues. The details of such a mechanism is standard (see e.g. [21]) and will not been explained in this paper.

The Pseudocode 5.1 shows the algorithm of the root agent. The computation starts with running of the root agent $ag^j,\ j \in K^1$. Its algorithm is presented in Pseudocode 5.1.

The root agent $ag^j,\ j \in K^1$ first generates the initial root deme using *GEN_ROOT*() function, then the metaepoch is run. Next, the agent performs sprout action activating one of the child agents. Here, the main loop of root agent is started.

Issuing an order of running the metaepoch by its children is followed by running its own metaepoch (if the status of the root agent is *active*).

After receiving from its children the confirmation that their metaepochs are finished, the sprout order for them is sent. Just like before, active root agent follows with sprouting here.

After receiving confirmations of finishing the sprout action by its children, the root agent checks their status—if there is no *inactive* child, the status of root is set to *stopped*. Now the global stopping condition is checked, if it is satisfied the computation is finished and all agents are stopped (an appropriate order is issued). Otherwise, the loop continues.

Pseudocode 5.2: CHILD AGENT ($ag^j : j \in K^{parent} \setminus K$) PSEUDOCODE

```
status ← inactive
finished ← false
parentidx ← PARENT(j)
while not finished
  b_receive(parentidx, order)
  switch order
    case ACTIVATE
      deme ← GEN_DEME(l(j), seed)
      status ← new
    case FINISH
      status ← inactive
      for each a ∈ GET_STATUS(j, {active})
        do send(a, FINISH)
      finished ← true
    case RUNMETA
      children ← GET_STATUS(j, {active, stopped, new})
      if not children = ∅
        then for each a ∈ children do send(j, RUNMETA)
      if status ∈ {new, active}
        then METAEPOCH()
      for each a ∈ children do b_receive(a, READY)
      if CHECK_STOP_COND()
        then status ← stopped
        else status ← active
      send(parentidx, READY)
    case RUNSPROUT
      if i < m − 1
        then for each a ∈ children
               do send(a, RUNSPROUT)
      if status = active
        then seed ← b_{l(j)}(deme)
             newagidx ← mind(deme, j)
             send(newagidx, seed, ACTIVATE)
             b_receive(newagidx, ACTIVATED)
      if i < m − 1
        then for each a ∈ children
               do b_receive(a, SPROUTED)
      if GET_STATUS(j, {inactive}) = ∅
        then status = stopped
      send(parentidx, SPROUTED)
```

Let us consider now the medium-level agent $ag^j,\ j \in K_{\text{par}} \setminus K^1$. Its algorithm is described in Pseudocode 5.2.

After initialising some basic attributes such as status or loop control variable, the medium-level agent starts working in a loop consisting of receiving and fulfilling the following orders sent from its parent:

- *ACTIVATE*—the deme is initialised and the status of the agent is set to *new*.
- *FINISH*—the agent stops the computation and sends appropriate message to all its children
- *RUNMETA*—the agent sends the same order to all its children then runs its own metaepoch if its status is *new* or *active*. Now the agent waits for the response from their children (*READY*). After checking the stopping condition the status of the agent may be changed. In the end *READY* message is sent to the parent.
- *RUNSPROUT*—if the children are not leaves of the agent tree, they are ordered by the agent to do *SPROUT* action. If the status of the agent is *active*, the sprout is performed. Now the agent waits for *SPROUTED* message from all its children. If there is no *inactive* children, the status of the agent is changed to *stopped*. Finally, *SPROUTED* message is sent to the parent.

Finally, the algorithm of the leaf-agents $\{ag^j\},\ j \in K^m$ is described in Pseudocode 5.3.

Pseudocode 5.3: LEAF AGENT ($ag^j,\ j \in K^m$) PSEUDOCODE

```
status ← inactive
finished ← false
parentidx ← PARENT(j)
while not finished
  b_receive(parentidx, order)
  switch order
    case ACTIVATE { deme ← GEN_DEME(m, seed)
                  { status ← new
    case FINISH   { status ← inactive
                  { finished ← true
    case RUNMETA  { if status ∈ {new, active}
                  {   then METAEPOCH()
                  { if CHECK_STOP_COND()
                  {   then status ← stopped
                  {   else status ← active
                  { send(parent, READY)
```

After initialising basic attributes such as status or loop control variable, the leaf agent starts working in a loop consisting of receiving and fulfilling the following orders sent from its parent:

- *ACTIVATE*—the deme is initialised and the status of the agent is set to *new*.
- *FINISH*—the agent stops the computation
- *RUNMETA*—the agent runs its metaepoch if its status is *new* or *active*. After checking the stopping condition the status of the agent may be changed. In the end *READY* message is sent to the parent.

After initialising basic attributes such as status or loop control variable, the leaf agent starts working in a loop consisting of receiving and fulfilling the following orders sent from its parent:

- *ACTIVATE*—the deme is initialised and the status of the agent is set to *new*.
- *FINISH*—the agent stops the computation.
- *RUNMETA*—the agent runs its metaepoch if its status is *new* or *active*. After checking the stopping condition the status of the agent may be changed. In the end *READY* message is sent to the parent.

6. Stochastic operators associated with the agents' actions

6.1. General structure of operators

Each action α^j of the agent $ag^j,\ j \in K$ can be represented as a pair of functions $(\delta^j_\alpha,\ \vartheta^j_\alpha)$. Function

$$\delta^j_\alpha : X \to \mathcal{M}(\{0, 1\}) \tag{30}$$

makes it possible to take the decision whether the action can be performed. The action α^j is performed with probability $\delta^j_\alpha(x)(1)$ by the agent ag^j at state $x \in X$ and rejected with probability $\delta^j_\alpha(x)(0)$.

Furthermore, the formula

$$\vartheta^j_\alpha : X \to \mathcal{M}(X) \tag{31}$$

defines non-deterministic state transition functions showing the effect of executing action α^j by the agent ag^j. The value of $\vartheta^j_\alpha(x)(y)$ denotes the probability of passing from the state x to y during the execution of α^j.

If any action is rejected, the trivial state transition

$$\vartheta_{\text{null}} : X \to \mathcal{M}(X), \qquad \vartheta_{\text{null}}(x)(y) = \begin{cases} 1 & \text{if } x = y \\ 0 & \text{otherwise} \end{cases} \tag{32}$$

is performed.

The probability transition function for action α^j performed by the agent ag^j, i.e. $\varrho^j_\alpha : X \to \mathcal{M}(X)$ is then defined by the formula

$$\varrho^j_\alpha(x)(y) = \delta^j_\alpha(x)(0) \cdot \vartheta_{\text{null}}(x)(y) + \delta^j_\alpha(x)(1) \cdot \vartheta^j_\alpha(x)(y), \tag{33}$$

where $x \in X$ denotes a current state and $y \in X$ is a consecutive state resulting from a conditional execution of α^j.

We will consider two type of actions:

- metaepoch actions $\{meta^j \mid j \in K\}$ available for all agents;
- sprouting actions $\{sprout^j \mid j \in K^{\mathrm{par}}\}$ defined only for the parental agents.

6.2. Metaepoch operators

Let us consider two consecutive states $x, y \in X$ appearing during the HGS computation. We will denote by (s^j, x^j) the components of x and by (z^j, y^j) the components of y.

We will start with the simplest one called $meta^j = (\delta^j_{meta}, \vartheta^j_{meta})$. If $j \in K^1$ (the root deme) we have

$$\delta^j_{meta}(x)(1) = \begin{cases} 1 & \text{if} \quad s^j = active, \\ 0 & \text{if} \quad s^j \neq active \end{cases} \tag{34}$$

and for $j \in K \setminus K^1$ (lower level demes)

$$\delta^j_{meta}(x)(1) = \begin{cases} 1 & \text{if} \quad s^j = active \text{ or } s^j = new, \\ 0 & \text{if} \quad s^j = inactive \text{ or } s^j = stopped, \end{cases} \tag{35}$$

and $\delta^j_{meta}(x)(0) = 1 - \delta^j_{meta}(x)(1)$. Now on the base of Observation 2.4, setting $i = l(j)$ we obtain

$$\vartheta^j_{meta}(x)(y) = \tau_i(x^j, y^j)\cdot$$
$$\begin{cases} S_i(x_j)\,[(x^j - y^j, f_i) < \mathrm{lsc}_i] & \text{if} \quad z^j = stopped \text{ and} \\ & \quad s^\eta = z^\eta,\ x^\eta = y^\eta,\ \forall \eta \in K \setminus \{j\}, \\ (1 - S_i(x_j))\,[(x^j - y^j, f_i) \geq \mathrm{lsc}_i] & \text{if} \quad z^j = active \text{ and} \\ & \quad s^\eta = z^\eta,\ x^\eta = y^\eta,\ \forall \eta \in K \setminus \{j\}, \\ 0 & \text{otherwise.} \end{cases} \tag{36}$$

6.3. Sprouting operators

Let us introduce several technical notions necessary for defining sprout operators. First, we consider the following family of probability distributions:

$$\begin{aligned} &\left\{\sigma_i^\xi\right\}_{\xi \in U_i,\ i=1,\ldots,m-1} \in \mathcal{M}(U_{i+1}); \\ &\sigma_i^\xi(U_{i+1}|\xi) = 1, \quad \forall \xi \in U_i,\ i = 1, \ldots, m-1 \end{aligned} \tag{37}$$

that are concentrated on the clusters $U_{i+1}|\xi$ (see (21)) (we will use it later for sprouting purposes).

Now we are able to define the family stochastic functions, that give us probability of creating the deme of individuals from $U_{i+1}|\xi$ by sampling according to the probability distribution σ_i^ξ. The probability of obtaining the arbitrary $y \in X_i$ may be computed using the Observation 2.1 as $\mathrm{Pr}_\rho^{\mu_{i+1}}(y),\ \rho = \sigma_i^\xi$.

Let us introduce a family of the following functions $\{b_i\},\ i = 1, \ldots, m-1$, such that:

$$\begin{aligned} &b_i : X_i \ni x \to \eta \in U_i; \\ &\eta = \min\{\zeta \in U_i;\ x_\zeta > 0,\ f_i(\zeta) \leq f_i(\gamma)\ \forall \gamma \in U_i\}. \end{aligned} \tag{38}$$

Each of these functions selects the best fitted individual η having the minimal genotype (according to an arbitrary order in U_i) from the deme with the vector x.

Finally, let us introduce the family of stochastic functions

$$\mathcal{T}_i : X_i \to \mathcal{M}(X_{i+1}); \qquad \mathcal{T}_i(x, y) = \mathrm{Pr}_\rho^{\mu_{i+1}}(y), \qquad \rho = \sigma_i^\xi, \qquad \xi = b_i(x), \quad i = 1, \ldots, m-1. \tag{39}$$

In order to compare an individual to a deme we define the functions $\{\mathcal{C}_i\}_{i=1,\ldots,m-1}$ such that

$$\mathcal{C}_i : U_i \times X_{i+1} \to \{0, 1\} \tag{40}$$

and $\mathcal{C}_i(\xi, x)$ returns 1 if the individual ξ is close enough in some sense to the deme represented by x and 0 otherwise. An important particular case of $\{\mathcal{C}_i\}_{i=1,\ldots,m-1}$ family checks the distance between the phenotype $\mathrm{code}_i(\xi)$ and the centre of the deme phenotypes $\frac{1}{\mu_{i+1}} \sum_{\xi \in U_{i+1}} x_\xi\, \mathrm{code}_{i+1}(\xi)$, so that

$$\mathcal{C}_i(\xi, x) = \begin{cases} 1 & \text{if} \quad \left\| \mathrm{code}_i(\xi) - \dfrac{1}{\mu_{i+1}} \displaystyle\sum_{\xi \in U_{i+1}} x_\xi\, \mathrm{code}_{i+1}(\xi) \right\| \leq \mathrm{dist}_i \\ 0 & \text{otherwise,} \end{cases} \tag{41}$$

where $\{\mathrm{dist}_i\},\ i = 1, \ldots, m-1$ are parameters of the family.

Next, let us define the sets of labels of child nodes of j *inactive* in the state $x \in X$.

$$\mathcal{I}_j^{\mathrm{in}}(x) = \left\{\eta \in \mathcal{I}_j | s^\eta = \mathit{inactive}\right\}, \tag{42}$$

where (s^η, x^η) is a component of the state $x \in X$ and $j \in K^{\mathrm{par}}$. Moreover, we define $\mathcal{I}_j^{\mathrm{as}}(x) = \mathcal{I}_j \setminus \mathcal{I}_j^{\mathrm{in}}(x)$. We will also need two functions

$$\begin{aligned} &a : X \times K^{\mathrm{par}} \ni (x, j) \to a(x, j) = \#\mathcal{I}_j^{\mathrm{in}}(x) \in \mathbb{N}, \\ &\mathit{mind} : X \times K^{\mathrm{par}} \to K; \quad \mathit{mind}(x, j) \in \mathcal{I}_j^{\mathrm{in}}(x) \\ &(\mathit{mind}(x, j))_{l(j)+1} = \min\left\{\eta_{l(j)+1} | \eta \in \mathcal{I}_j^{\mathrm{in}}(x)\right\}. \end{aligned} \tag{43}$$

Now we are ready for defining the decision functions and the Markov kernels for the sprout operations of demes indexed by $j \in K^{\mathrm{par}}$;

$$\delta^j_{\mathit{sprout}}(x)(1) = \begin{cases} 1 & \text{if} \quad s^j = \mathit{active} \text{ and} \\ & \quad \mathcal{C}_i(\zeta, x^\eta) = 0 \ \forall \eta \in \mathcal{I}_j^{\mathrm{as}}(x), \\ & \quad \text{for } \zeta = b_i(x^j), \\ 0 & \quad \text{otherwise} \end{cases} \tag{44}$$

and $\delta^j_{\mathit{sprout}}(x)(0) = 1 - \delta^j_{\mathit{sprout}}(x)(1)$. Setting $i = l(j)$ we have:

$$\vartheta^j_{\mathit{sprout}}(x)(y) = \begin{cases} \mathcal{T}_i(x^j, w) & \text{if} \quad \left(\left(z^j = \mathit{active} \text{ and } a(x, j) > 1\right) \text{ or}\right. \\ & \quad \left.\left(z^j = \mathit{stopped} \text{ and } a(x, j) = 1\right)\right) \text{ and} \\ & \quad z^{\mathit{mind}(x,j)} = \mathit{new} \text{ and } y^{\mathit{mind}(x,j)} = w \text{ and} \\ & \quad s^\eta = z^\eta, \ y^\eta = x^\eta, \ \forall \eta \in K, \eta \neq \mathit{mind}(x, j), \\ 0 & \quad \text{otherwise.} \end{cases} \tag{45}$$

7. The transition probability function for the whole system

Let us take two arbitrary Markov kernels

$$\varrho^1, \ \varrho^2 : X \to \mathcal{M}(X). \tag{46}$$

Their composition $\varrho^2 \circ \varrho^1 : X \to \mathcal{M}(X)$ can be computed using the following formula

$$\varrho^2 \circ \varrho^1(x)(y) = \sum_{u \in X} \varrho^1(x)(u)\, \varrho^2(u)(z), \quad \forall x, y \in X. \tag{47}$$

In the sequel we shall show that for some actions α^j, $\alpha^{j'}$ of the same type α their kernels commute, i.e.

$$\varrho^j_\alpha \circ \varrho^{j'}_\alpha = \varrho^{j'}_\alpha \circ \varrho^j_\alpha,$$

which means that the effect of the actions' execution is independent upon the order of the execution. And this in turn means that the actions do not need mutual synchronisation, therefore they can be safely performed in parallel.

We shall start from defining a property of actions especially useful in the context of the commutativity.

Definition 7.1. Let L be a nonempty subset of K. We say that a kernel $\varrho : X \to \mathcal{M}(X)$ is *localised in* L if

(1) it does not change components of the system state describing demes from outside L, i.e. for any $x, y \in X$ such that there exists $j \in K \setminus L$ with $(s^j, x^j) \neq (z^j, y^j)$ we have

$$\varrho(x)(y) = 0; \tag{48}$$

(2) it does not depend upon components of the system state describing demes from outside L, i.e. for any $x, \bar{x}, y, \bar{y} \in X$ equalities

$$(s^j, x^j) = (z^j, y^j), \qquad (\bar{s}^j, \bar{x}^j) = (\bar{z}^j, \bar{y}^j) \quad \text{for } j \notin L$$

and

$$(s^j, x^j) = (\bar{s}^j, \bar{x}^j), \qquad (z^j, y^j) = (\bar{z}^j, \bar{y}^j) \quad \text{for } j \in L$$

imply that

$$\varrho(x)(y) = \varrho(\bar{x})(\bar{y}). \tag{49}$$

The usefulness of the locality property is shown by the following proposition.

Proposition 7.1. *Any two kernels localized in disjoint sets of demes commute, i.e. for every ϱ^1 localized in L_1 and ϱ^2 localized in L_2 with $L_1 \cap L_2 = \emptyset$ we have*

$$\varrho^2 \circ \varrho^1 = \varrho^1 \circ \varrho^2.$$

Proof. Take arbitrary $x, y \in X$. Let us compute the composition $\varrho^2 \circ \varrho^1$. Namely

$$\varrho^2 \circ \varrho^1(x)(y) = \sum_{u \in X} \varrho^1(x)(u)\varrho^2(u)(y). \tag{50}$$

Thanks to (48) the sum is in fact taken over such $u = (t^j, u^j)_{j \in K}$ that

$$(t^j, u^j) = (s^j, x^j) \quad \text{for } j \in K \setminus L_1 \tag{51}$$

$$(t^j, u^j) = (z^j, y^j) \quad \text{for } j \in K \setminus L_2. \tag{52}$$

But since $L_1 \cap L_2 = \emptyset$ we have that $L_1 \subset K \setminus L_2$ and $L_2 \subset K \setminus L_1$. Hence either $(s^j, x^j) \neq (z^j, y^j)$ for some $j \in K \setminus (L_1 \cup L_2)$ and then both sides of (50) are equal to 0, or the only u satisfying conditions (51) is given by the following formula

$$(t^j, u^j) = \begin{cases} (s^j, x^j) & \text{for } j \in L_2 \\ (z^j, y^j) & \text{for } j \in L_1 \\ (s^j, x^j) = (z^j, y^j) & \text{for } j \in K \setminus (L_1 \cup L_2). \end{cases}$$

Furthermore, if we take $\overline{u} \in X$ such that $(\overline{t}^j, \overline{u}^j) = (z^j, y^j)$ for $j \in L_2$ (and hence $j \notin L_1$) and $(\overline{t}^j, \overline{u}^j) = (s^j, x^j)$ for $j \notin L_2$, then from (49) we obtain

$$\varrho^2(u)(y) = \varrho^2(x)(\overline{u}).$$

Thus,

$$\varrho^2 \circ \varrho^1(x)(y) = \varrho^1(x)(u)\varrho^2(x)(\overline{u}).$$

Exactly the same result is obtained when we compute $\varrho^1 \circ \varrho^2(x)(y)$. □

Proposition 7.1 is the tool for showing the commutativity of the vast part of considered actions. It is performed in the following series of propositions and corollaries.

Proposition 7.2. *Action meta^j is localised in $\{j\}$.*

Proof. It is a straightforward consequence of the definition of δ^j_{meta}, ϑ^j_{meta} and ϑ_{null}, as we note that δ^j_{meta} depends only on s_j, ϑ^j_{meta} depends only on x^j and it changes only s^j and x^j. ϑ_{null} does not depend on anything and does not change anything. □

Corollary 7.3. *Kernels of metaepoch actions performed in different demes commute, i.e. for $j, j' \in K$ such that $j \neq j'$ we have*

$$\varrho^j_{meta} \circ \varrho^{j'}_{meta} = \varrho^{j'}_{meta} \circ \varrho^j_{meta}.$$

Proposition 7.4. *Action sprout^j is localised in $\{j\} \cup \mathcal{U}_j$.*

Proof. Condition (48) is easily verified because ϑ^j_{sprout} changes at most components $s^j, s^{mind(x,j)}$ and $x^{mind(x,j)}$ with $mind(x, j) \in \mathcal{U}_j$. Similarly, to prove (49) it is enough to note that δ^j_{sprout} depends only on s^j, x^j and x^η for $\eta \in \mathcal{U}^{\text{as}}_j(x) \subset \mathcal{U}_j$, whereas ϑ^j_{sprout} depends only on x^j and s^η for $\eta \in \mathcal{U}_j$. □

Corollary 7.5. *Kernels of sprouting actions performed in different demes such that neither of them is the parent of the other commute, i.e. for $j, j' \in K$ such that $j \neq j'$, $j' \notin \mathcal{U}_j$ and $j \notin \mathcal{U}_{j'}$ we have*

$$\varrho^j_{sprout} \circ \varrho^{j'}_{sprout} = \varrho^{j'}_{sprout} \circ \varrho^j_{sprout}.$$

The following proposition fills the gap. It is, however, not a consequence of Proposition 7.1 and it has to be proved in a different way.

Proposition 7.6. *Kernels of sprouting actions performed in different demes such that one of them is the parent of the other commute, i.e. for $j, j' \in K$ such that $j \neq j'$, $j' \in \mathcal{U}_j$ and $j \in K^{\text{par}}$ we have*

$$\varrho^j_{sprout} \circ \varrho^{j'}_{sprout} = \varrho^{j'}_{sprout} \circ \varrho^j_{sprout}.$$

Proof. Take arbitrary $x, y \in X$. We have to compute

$$\varrho^{j'}_{sprout} \circ \varrho^j_{sprout}(x)(y) = \sum_{u \in X} \varrho^{j'}_{sprout}(u)(y)\varrho^j_{sprout}(x)(u).$$

Let us start by noting that if $j' = mind(x, j)$, then during $sprout^j$ it changes its status from *inactive* to *new*. In neither of them it may sprout, hence it is forced to perform null action, which is trivially commutative with any other. Thus, without the loss of generality we can assume that $j' \neq mind(x, j)$. Similarly, when the conditions of the performability of *sprout* are not satisfied in j or j' we obtain at least one null term, so the commutativity becomes trivial. To sum up, the only interesting case is when

$$\begin{aligned}
&j' \in \mathcal{I}_j, \quad j' \neq mind(x, j), \qquad s^j = s^{j'} = active\\
&\mathcal{C}_{l(j)}\left(b_{l(j)}(x^j), x^\eta\right) = 0 \quad \text{for } \eta \in \mathcal{I}_j^{\mathrm{as}}(x)\\
&\mathcal{C}_{l(j')}\left(b_{l(j')}(x^{j'}), x^\eta\right) = 0 \quad \text{for } \eta \in \mathcal{I}_{j'}^{\mathrm{as}}(x).
\end{aligned}$$

In this case the decision of performing $sprout^j$ is obviously positive. Moreover, from (45) we have that

$$\vartheta^j_{sprout}(x)(u) = \mathcal{T}_{l(j)}(x^j, w)$$

for $t^j = active$ or *stopped* (depending on $a(x, j)$), $t^{mind(x,j)} = new$, $u^{mind(x,j)} = w$ and $(t^\eta, u^\eta) = (s^\eta, x^\eta)$ for $\eta \neq mind(x, j)$. In particular $t^{j'} = s^{j'} = active$ and $u^{j'} = x^{j'}$. As a consequence

$$\mathcal{C}_{l(j')}\left(b_{l(j')}(u^{j'}), x^\eta\right) = \mathcal{C}_{l(j')}\left(b_{l(j')}(x^{j'}), x^\eta\right) = 0$$

for $\eta \in \mathcal{I}_{j'}^{\mathrm{as}}(u) = \mathcal{I}_{j'}^{\mathrm{as}}(x)$. Hence, also the decision of performing $sprout^{j'}$ is positive. Moreover, $\mathcal{I}_{j'}^{\mathrm{in}}(u) = \mathcal{I}_{j'}^{\mathrm{in}}(x)$, thus $a(u, j') = a(x, j')$ and $mind(u, j') = mind(x, j')$. Hence, we obtain

$$\vartheta^{j'}_{sprout}(u)(y) = \mathcal{T}_{l(j')}(u^{j'}, w') = \mathcal{T}_{l(j')}(x^{j'}, w')$$

for $z^{j'} = active$ or *stopped* (depending on $a(x, j')$), $z^{mind(x,j')} = new$,
$y^{mind(x,j')} = w'$ and $(z^\eta, y^\eta) = (t^\eta, u^\eta)$ for $\eta \neq mind(x, j')$. To sum up, the only components of y that are not uniquely determined are $y^{mind(x,j)}$ and $y^{mind(x,j')}$. Hence, we obtain

$$\sum_{u\in X} \varrho^{j'}_{sprout}(u)(y)\varrho^{j}_{sprout}(x)(u) = \sum_{w\in X_{l(j)}} \sum_{w'\in X_{l(j')}} \mathcal{T}_{l(j)}(x^j, w)\mathcal{T}_{l(j')}(x^{j'}, w').$$

The same result is obtained when we compute $\varrho^{j}_{sprout} \circ \varrho^{j'}_{sprout}(x)(y)$. □

Corollary 7.7. *Kernels of any two sprouting actions performed in different parental demes commute, i.e. for $j, j' \in K^{\mathrm{par}}$ such that $j \neq j'$ we have*

$$\varrho^{j}_{sprout} \circ \varrho^{j'}_{sprout} = \varrho^{j'}_{sprout} \circ \varrho^{j}_{sprout}.$$

The agent based synchronisation scheme ensures, that the metaepoch steps are completed by all active demes before the sprouts are activated (see Section 5). The transition probability function for the metaepoch step $\tau_{meta} : X \to \mathcal{M}(X)$ is the composition of Markov kernels ϱ^j_{meta}, $j \in K$, so

$$\tau_{meta} = \bigcirc_{j\in K} \varrho^j_{meta}, \tag{53}$$

where $\bigcirc$ denotes the group composition operator. The order of composition (53) is unimportant (see Corollary 7.3). Similarly, the transition probability function for the sprout step $\tau_{sprout} : X \to \mathcal{M}(X)$ is the composition of Markov kernels ϱ^j_{sprout}, $j \in K^{\mathrm{par}}$, so

$$\tau_{sprout} = \bigcirc_{j\in K^{\mathrm{par}}} \varrho^j_{sprout}, \tag{54}$$

where again the order of composition is unimportant (see Corollary 7.7).

The transition probability function $\tau : X \to \mathcal{M}(X)$ for the whole system is the composition imposed by metaepoch and sprout steps, so

$$\tau = \tau_{sprout} \circ \tau_{meta}. \tag{55}$$

8. Conclusions

The families of stochastic operators $\{\tau_i\}_{i=1,\ldots,m}$, $\{\mathcal{T}_i\}_{i=1,\ldots,m-1}$, and deterministic functions $\{\mathcal{C}_i\}_{i=1,\ldots,m-1}$, (see (25), (39), (40)) constitute the core of the stochastic dynamic of the HGS strategy. We delivered their important case based on the Vose's concept of SGA [16] in Section 2. The presented stochastic model can be extended to other cases of encoding and genetic operations by redefining these operators.

The agent based scheme presented in this paper introduced minimum synchronisation that allows to express HGS behaviour by the single Markov chain dynamics. More relaxed synchronisation scheme that allows for independent branch development was presented in [22].

Agent-based synchronisation mechanism allows to further enhance the system in order to introduce inter-agent information exchange resulting, e.g., in adaptation of parameters of local variation operators, following the trends observed in the neighbouring agents. Agents may also participate in enhancing the technical level of the computation, taking care of, e.g., load balancing on the computational nodes.

The proposed pseudocode describes a system that may be easily implemented after performing more full analysis and design. The required matters that were not covered in the pseudocode are, e.g., data structures for holding the information about HGS tree, the agent's life-cycle (initiation of all agents in the beginning of computation should be replaced by dynamic creation and disposal, etc.).

Therefore, two main goals of the presented pseudocode are: explanation of the actual algorithms of the system synchronisation and making possible maintaining Markov-conditions (dependence of the subsequent state only on the last perceived one) for the verification of the asymptotic features of the model.

The transition probability function for the whole system was defined along with sketching the proofs of commutativity of *sprout* and *metaepoch* actions. These proofs are necessary to allow Markov-chain modelling of this parallel system.

The proposed synchronisation mechanism allows easy implementation of the system in distributed computational environments because of easy, scalable communication structure (along inter-agent tree edges).

Acknowledgement

The work has been partially supported by the Polish Ministry of Science and Higher Education grant no. NN 519 447739.

References

[1] A. Byrski, R. Schaefer, Stochastic model of evolutionary and immunological multi-agent systems: mutually exclusive actions, Fundamenta Informaticae 95 (2–3) (2009) 263–285.
[2] R. Schaefer, A. Byrski, M. Smołka, Stochastic model of evolutionary and immunological multi-agent systems: parallel execution of local actions, Fundamenta Informaticae 95 (2–3) (2009) 325–348.
[3] M. Carvalho, C. Perez, An evolutionary multi-agent approach to anomaly detection and cyber defense, in: CSIIRW'11: Proceedings of the 7th Annual Workshop on Cyber Security and Information Intelligence Research, ACM, 2011.
[4] W.D. Whitley, S.B. Rana, R.B. Heckendorn, Island model genetic algorithms and linearly separable problems, in: Selected Papers from AISB Workshop on Evolutionary Computing, Springer-Verlag, London, UK, 1997, pp. 109–125.
[5] R. Schaefer, J. Kołodziej, Genetic search reinforced by the population hierarchy, in: K.D. Jong, et al. (Eds.), Foundations of Genetic Algorithms VII, Morgan Kaufmann, 2003, pp. 369–385.
[6] J. Kołodziej, W. Jakubiec, M. Starczak, R. Schaefer, Hierarchical genetic strategy applied to the problem of the coordinate measuring machine geometrical errors, in: T. Burczyński, et al. (Eds.), IUTAM'02 Symposium on Evolutionary Methods in Mechanics, 24–27 September 2002, Kluwer Ac. Press, Cracow, Poland, 2004, pp. 22–30.
[7] J. Kołodziej, F. Xhafa, Enhancing the genetic-based scheduling in computational grids by a structured hierarchical population, Future Generation Computer Systems 27 (2011) 1035–1045.
[8] N.N. Schraudolph, R.K. Belew, Dynamic parameter encoding for genetic algorithms, Machine Learning 9 (1992) 9–21.
[9] D. Whitley, K. Mathias, P. Fitzhorn, Delta coding: an iterative search strategy for genetic algorithms, in: R. Belew, L. Booker (Eds.), Proc. of the 4th International Conference on Genetic Algorithms, Morgan Kaufmann, 1991, pp. 77–84.
[10] M. Srinivas, L. Patnaik, Adaptive probabilities of crossover and mutation in genetic algorithms, IEEE Transactions on Systems, Man and Cybernetics 24 (1994) 656–667.
[11] J. Zhang, H. Chung, W. Lo, Clustering-based adaptive crossover and mutation probabilities for genetic algorithms, IEEE Transactions on Evolutionary Computation 11 (2007) 326–335.
[12] X. Yueju, L. Shuguang, Y. Jingfeng, C. Qiang, Genetic algorithm based adaptive neural network ensemble and its application in predicting carbon flux, in: Proc. of the 3rd International Conference on Natural Computation, ICNC 207, IEEE Press, 2007, pp. 183–187.
[13] R. Schaefer, H. Telega, Foundation of Global Genetic Optimization, in: Studies in Computational Intelligence, vol. 74, Springer Verlag, Berlin, Heidelberg, New York, 2007.
[14] A.E. Nix, M.D. Vose, Modeling genetic algorithms with Markov chains, Annals of Mathematics and Artificial Intelligence 5 (1992) 79–88.
[15] L.M. Schmitt, Theory of genetic algorithm, Theoretical Computer Science 259 (2001) 1–61.
[16] M. Vose, The Simple Genetic Algorithm: Foundations and Theory, MIT Press, Cambridge, MA, USA, 1998.
[17] P. Billingsley, Probability and Measure, Wiley-Interscience, 1995.
[18] K. Cetnarowicz, M. Kisiel-Dorohinicki, E. Nawarecki, The application of evolution process in multi-agent world (MAW) to the prediction system, in: M. Tokoro (Ed.), Proc. of the 2nd Int. Conf. on Multi-Agent Systems, ICMAS'96, AAAI Press, 1996.
[19] A. Byrski, R. Schaefer, M. Smołka, C. Cotta, Asymptotic analysis of computational multi-agent systems, in: R. Schaefer, C. Cotta, J. Kołodziej, G. Rudolph (Eds.), Proceedings of 11th International Conference on Parallel Problem Solving from Nature—PPSN XI, in: LNCS, vol. 6238, Springer-Verlag, 2011, pp. 475–484.
[20] A. Byrski, M. Kisiel-Dorohinicki, M. Carvalho, A crisis management approach to mission survivability in computational multi-agent systems, Computer Science 11 (2010) 99–113.
[21] A. Silberschatz, P.B. Galvin, G. Gagne, Operating System Concepts with Java, eighth ed., Wiley, 2010.
[22] J. Momot, K. Kosacki, M. Grochowski, P. Uhruski, R. Schaefer, Multi-agent system for irregular parallel genetic computations, in: W.I. Grosky, F. Plasil (Eds.), SofSem 2002: Theory and Practice of Informatics, in: LNCS, vol. 3038, Springer Verlag, 2004, pp. 623–630.