

The Genetic Algorithm of GMSE

GMSE: an R package for generalised management strategy evaluation (Supporting Information 1)

A. Bradley Duthie^{1,3}, Jeremy J. Cusack¹, Isabel L. Jones¹, Jeroen Minderman¹, Erlend B. Nilsen², Rocío A. Pozo¹, O. Sarobidy Rakotonarivo¹, Bram Van Moorter², and Nils Bunnefeld¹

[1] *Biological and Environmental Sciences, University of Stirling, Stirling, UK* [2] *Norwegian Institute for Nature Research, Trondheim, Norway* [3] *alexander.duthie@stir.ac.uk*

Extended introduction to the genetic algorithm applied in GMSE

Game theory is the formal study of strategic interactions, and can therefore be applied to modelling stakeholder actions and addressing issues of cooperation and conflict in conservation (Lee, 2012; Kark et al., 2015; Adami et al., 2016; Tilman et al., 2016; Redpath et al., 2018). In game-theoretic models, agents adopt strategies to make decisions that maximise some type of payoff (e.g., utility, biological fitness). Agents are constrained in their decision-making, and realised pay-offs depend on decisions made by other agents. In simple models, it is often useful to assume that agents are perfectly rational decision-makers, then find optimal solutions for pay-off maximisation mathematically. But models that permit even moderately complex decision-making strategies or pay-off structures often include more possible strategies than are mathematically tractable (Hamblin, 2013). In these models, genetic algorithms, which mimic the process of natural selection (mutation, recombination, selection, reproduction), can find adaptive (i.e., practical, but not necessarily optimal) solutions for game strategies (e.g., Balmann and Happe, 2000; Tu et al., 2000; Hamblin, 2013).

A genetic algorithm is called in the predefined GMSE manager and user models to simulate human decision making. As of GMSE version 0.4.0.3, this includes one independent call to the genetic algorithm for each decision-making agent in every GMSE time step. Therefore, one run of the genetic algorithm occurs to simulate the manager’s policy-setting decisions in each time step (unless otherwise defined through non-default `manage_freq` values greater than 1; e.g., see SI6), and one run occurs to simulate each individual user’s action decisions in each time step (unless otherwise defined through non-default `group_think = TRUE`, in which case one user makes decisions that all other users copy). Each run of the genetic algorithm mimics the evolution by natural selection of a population of potential manager or user strategies over multiple iterations, with the highest fitness strategy in the terminal iteration being selected as the one that the manager or user decides to implement. For clarity, as in the main text, we use ‘time step’ to refer to a full GMSE cycle (in which multiple genetic algorithms may be run) and ‘iteration’ to refer to a single, non-overlapping, generation of potential strategies that evolve within a genetic algorithm (see Figure 1 of the main text). Below, we explain the genetic algorithm in detail, as it occurs in GMSE v0.4.0.7 (future versions of GMSE might expand upon this framework, and we highlight some of these potential avenues for expansion). We first explain the key data structures used, then provide an overview of how a population of strategies is initialised, and the subsequent processes of crossover, mutation, cost constraint, fitness evaluation, tournament selection, and replacement. We then explain the fitness functions of managers and users in more detail.

Key data structures used

The focal data structure used for tracking manager and user decisions is a three dimensional array, which we will call ACTION (also returned as `user_array` by `gmse_apply`; see SI7). Rows of ACTION correspond to the entities affected by actions (resources, landscape properties, or potentially other agents), and columns correspond either to properties of the affected entities, or to the actions potentially allocated to them. Each

43 layer of ACTION corresponds to a unique agent, the first of which is the manager; additional layers correspond
 44 to users. Below shows an ACTION array for a GMSE model with one manager and two users.

```

45 ## , , Manager_Actions
46 ##
47 ##          Act Type_1 Type_2 Type_3      Util. U_land U_loc.  Score Cull
48 ## Resource   -2     1     0     0 1000.00000      0     0     0     0
49 ## Landscape  -1     1     0     0   0.00000      0     0     0     0
50 ## Res_cost    1     1     0     0 -20.40816      0     0    10    58
51 ## U1_cost     2     1     0     0   0.00000      0     0     0     0
52 ## U2_cost     3     1     0     0   0.00000      0     0     0     0
53 ##          Castrate Feed Help_off None
54 ## Resource      0     0         0     0
55 ## Landscape     0     0         0     0
56 ## Res_cost     10    10         10    62
57 ## U1_cost       0     0         0     0
58 ## U2_cost       0     0         0     0
59 ##
60 ## , , User_1_Actions
61 ##
62 ##          Act Type_1 Type_2 Type_3 Util. U_land U_loc.  Score Cull Castrate
63 ## Resource   -2     1     0     0  -1      0     0     0    17     0
64 ## Landscape  -1     1     0     0   0      0     0     0     0     0
65 ## Res_cost    1     1     0     0   0      0     0     0     0     0
66 ## U1_cost     2     1     0     0   0      0     0     0     0     0
67 ## U2_cost     3     1     0     0   0      0     0     0     0     0
68 ##          Feed Help_off None
69 ## Resource      0         0     0
70 ## Landscape     0         0     0
71 ## Res_cost      0         0     0
72 ## U1_cost       0         0     0
73 ## U2_cost       0         0     0
74 ##
75 ## , , User_2_Actions
76 ##
77 ##          Act Type_1 Type_2 Type_3 Util. U_land U_loc.  Score Cull Castrate
78 ## Resource   -2     1     0     0  -1      0     0     0    17     0
79 ## Landscape  -1     1     0     0   0      0     0     0     0     0
80 ## Res_cost    1     1     0     0   0      0     0     0     0     0
81 ## U1_cost     2     1     0     0   0      0     0     0     0     0
82 ## U2_cost     3     1     0     0   0      0     0     0     0     0
83 ##          Feed Help_off None
84 ## Resource      0         0     0
85 ## Landscape     0         0     1
86 ## Res_cost      0         0     0
87 ## U1_cost       0         0     0
88 ## U2_cost       0         0     0

```

89 The above array holds all of the information on manager and user actions. The first seven columns contain
 90 information about which entities are affected, and how they are affected. The first column **Act** identifies
 91 the type of action being performed; a value of -2 defines a direct action to a resource (e.g., culling of the
 92 resource), and a value of -1 defines direct action to a landscape (e.g., increasing yield). Positive values are
 93 currently only meaningful for **Manager_Actions**, where a value of 1 defines an action setting a uniform cost
 94 of users' direct actions on resources (i.e., costs where **Act** = -2 for **User_1_Actions** and **User_2_Actions**).
 95 All other values for **Act** are meaningless in GMSE 0.4.0.3, but might be expanded upon in future versions

96 to allow for modification of specific user costs enacted by managers (i.e., managers having different policies
 97 for different users) or other users (e.g., users increasing the costs of other users' actions due to conflict or
 98 cooperation). We will therefore focus only on rows 1-3 of ACTION.

99 Columns 2-4 refer to resource or landscape types, but only Type_1 = 1, Type_2 = 0, and Type_3 = 0
 100 are allowed in predefined GMSE v0.4.0.7 manager and user sub-models (i.e., only one type of resource is
 101 permitted). Future versions might allow for different resource types (e.g., Type_1 might be used to designate
 102 species, and Type_2 and Type_3 could designate stage or sex). Column 5 Util. of ACTION defines the utility
 103 associated with the resource (where Act = -2) or landscape (where Act = -1). For managers, the target
 104 resource abundance set with the GMSE argument manage_target is found in row 1 (1000 in ACTION above);
 105 for users, the value in row 1 identifies whether resources are preferred to increase (if positive) or decrease (if
 106 negative). Values of column 5 in row 2 similarly identify whether landscape cell output is preferred by users
 107 to increase or decrease (managers do not currently have preferences for landscape output). Of special note is
 108 row 3 for Manager_Actions, which defines the current manager's utility for resources; that is, the adjustment
 109 to resource abundance that the manager will attempt to make based on the manage_target and the most
 110 recent estimate of resource abundance produced by the observation model (in the case of the above, resource
 111 abundance is estimated at ca 1020.41, so the manager will set policy in attempt to change the population
 112 size by ca -20.41 resources). Column 6 U_land defines whether or not the utility attached to the resource or
 113 landscape output depends on it being on a landscape cell that is owned by the acting user. Related, column
 114 7 U_loc. defines whether or not actions can be performed only on a landscape cell that is owned by the
 115 acting user. Hence values of columns 6 and 7 are binary, and affected by the land_ownership argument
 116 in gmse and gmse_apply. Finally, columns 8-13 correspond to specific actions, either direct (where Act <
 117 0) or indirect by setting policy (for row 3 of Manager_Actions where Act = 1). The last column 13 None
 118 corresponds with no actions. See GMSE documentation for details about the effects of each action.

119 Constraints on the values that elements in the ACTION array can take are defined by a COST array (also
 120 returned as manager_array by gmse_apply; see SI7) of dimensions identical to ACTION. Elements of COST
 121 define how many units from the manager_budget or user_budget are needed to perform a single action; a
 122 minimum_cost for actions is defined as an argument in GMSE (10 by default). All values in COST columns 1-7
 123 are set to 100001, one higher than the highest possible manager_budget or user_budget, so neither managers
 124 nor users can affect resource types or utilities. Columns 8-13 are also set to 10001, except where actions are
 125 allowed. Maximum values of 100000 are independent of any other parameter value specified in GMSE (e.g.,
 126 landscape dimensions). Below shows the COST array that corresponds to the above ACTION array.

```

127 ## , , Manager_Costs
128 ##
129 ##           Act Type_1 Type_2 Type_3  Util. U_land U_loc.  Scare  Cull
130 ## Resource  100001 100001 100001 100001 100001 100001 100001 100001 100001
131 ## Landscape 100001 100001 100001 100001 100001 100001 100001 100001 100001
132 ## Res_cost  100001 100001 100001 100001 100001 100001 100001 100001 10
133 ## U1_cost   100001 100001 100001 100001 100001 100001 100001 100001 100001
134 ## U2_cost   100001 100001 100001 100001 100001 100001 100001 100001 100001
135 ##           Castrate  Feed Help_off  None
136 ## Resource  100001 100001 100001 10
137 ## Landscape 100001 100001 100001 10
138 ## Res_cost  100001 100001 100001 10
139 ## U1_cost   100001 100001 100001 100001
140 ## U2_cost   100001 100001 100001 100001
141 ##
142 ## , , User_1_Costs
143 ##
144 ##           Act Type_1 Type_2 Type_3  Util. U_land U_loc.  Scare  Cull
145 ## Resource  100001 100001 100001 100001 100001 100001 100001 100001 58
146 ## Landscape 100001 100001 100001 100001 100001 100001 100001 100001 100001
147 ## Res_cost  100001 100001 100001 100001 100001 100001 100001 100001 100001

```

```

148 ## U1_cost    100001 100001 100001 100001 100001 100001 100001 100001 100001 100001
149 ## U2_cost    100001 100001 100001 100001 100001 100001 100001 100001 100001 100001
150 ##           Castrate   Feed Help_off   None
151 ## Resource    100001 100001   100001    10
152 ## Landscape   100001 100001   100001    10
153 ## Res_cost    100001 100001   100001 100001
154 ## U1_cost     100001 100001   100001 100001
155 ## U2_cost     100001 100001   100001 100001
156 ##
157 ## , , User_2_Costs
158 ##
159 ##           Act Type_1 Type_2 Type_3 Util. U_land U_loc.  Scare  Cull
160 ## Resource    100001 100001 100001 100001 100001 100001 100001 100001 58
161 ## Landscape   100001 100001 100001 100001 100001 100001 100001 100001 100001
162 ## Res_cost    100001 100001 100001 100001 100001 100001 100001 100001 100001
163 ## U1_cost     100001 100001 100001 100001 100001 100001 100001 100001 100001
164 ## U2_cost     100001 100001 100001 100001 100001 100001 100001 100001 100001
165 ##           Castrate   Feed Help_off   None
166 ## Resource    100001 100001   100001    10
167 ## Landscape   100001 100001   100001    10
168 ## Res_cost    100001 100001   100001 100001
169 ## U1_cost     100001 100001   100001 100001
170 ## U2_cost     100001 100001   100001 100001

```

171 Note that in default GMSE parameters, `culling = TRUE`, but all other actions are set to `FALSE`. Hence, the
172 `Cull` column 9 is the only column besides column 13 `None` in which cost is less than 100001. Manager’s
173 actions in `ACTION` directly affect the cost of users performing one of the five possible actions on resources
174 (columns 8-12). This can be verified in `ACTION` where the manager has set the cost of culling to 58 (row 3),
175 and the corresponding `COST` of resource culling is 58 for both users (row 1). The cost of the manager affecting
176 the cost of user actions is always set to the `minimum_cost`; here the default 10 is used. This `minimum_cost`
177 also defines cost values for `None`, in which the user or manager does nothing, as might occur if the manager
178 wants to permit culling and therefore does not want to invest any of their `manager_budget` to increasing the
179 cost of culling. Both `ACTION` and `COST` are updated in each time step unless `manage_freq > 1`, in which case
180 `COST` and `Manager_Actions` in `ACTION` are updated at the frequency defined.

181 General overview of key aspects of the genetic algorithm

182 The genetic algorithm updates a single layer of the `ACTION` array, which defines the decisions of a single agent
183 (either the manager or a user). The corresponding layer of the `COST` array remains unchanged, and serves
184 only to ensure that `ACTION` values do not exceed `manager_budget` or `user_budget` for managers and users,
185 respectively. The genetic algorithm proceeds by first initialising a large (but temporary) population of new
186 `ACTION` layers. In each iteration, these layers crossover and mutate, generating variation in potential agent
187 decisions; costs constrain this variation from exceeding a maximum budget, then the fitness of each layer is
188 evaluated based on how the layer is predicted to affect resources or landscape output to which the agent has
189 assigned some utility. A tournament is used to select high fitness layers, and these selected layers become the
190 new iteration of layers; iterations continue until a minimum number of iterations (`ga_mingen`) have passed
191 and a convergence criteria is satisfied such that the increase in mean fitness from the previous iteration is
192 below the threshold `converge_crit` (Figure 1 below).

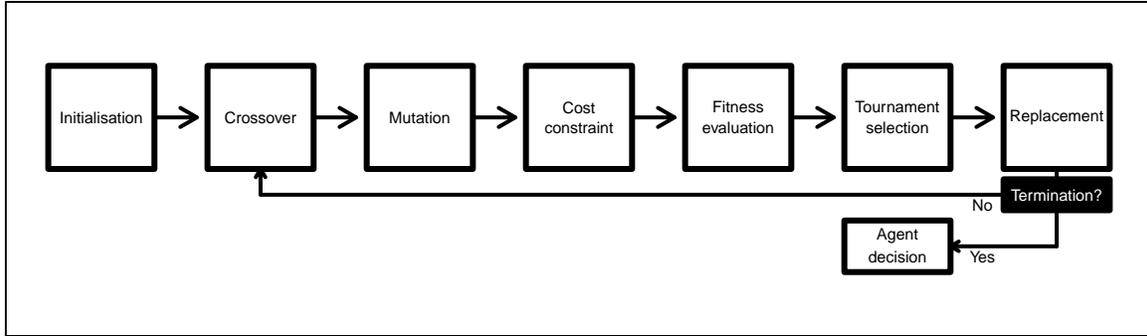


Figure 1: Conceptual overview of the GMSE genetic algorithm

193 Initialisation

194 At the start of each genetic algorithm, a population of size `ga_popsiz` is initialised (hereafter the `POPULATION`
 195 array). This population is held in a 3D array of `ga_popsiz` layers. Each layer includes an identical number
 196 of rows and columns as in `ACTION`, and one layer defines a single ‘individual’ in the population. The first seven
 197 columns of `ACTION` are replicated exactly for all individuals, and remain unchanged throughout the genetic
 198 algorithm thereby preserving the information about which entities are affected by actions in a given row. The
 199 remaining columns are either also replicated exactly as in `ACTION` (i.e., initialised to be the same decisions
 200 as in a previous time step), or randomly seeded with values given the constraints of `manager_budget` or
 201 `user_budget` (i.e., initialised to random decision making). The number of exact replicates initialised is set
 202 using `ga_seedrep` (if `ga_seedrep` \geq `ga_popsiz`, then all individuals are seeded as replicates). After the
 203 `POPULATION` of `ga_popsiz` individuals is initialised, a loop simulating the adaptive evolution of `POPULATION`
 204 in non-overlapping iterations begins (see Figure 1 above).

205 Crossover

206 A single iteration of the genetic algorithm begins with a uniform crossover (Hamblin, 2013), by which actions
 207 of individuals in `POPULATION` are randomly swapped with some probability. To implement crossover, each
 208 individual selects a partner, then exchanges corresponding array elements affecting agent actions (columns
 209 8-13) with their partner at a fixed probability of `ga_crossover`.

210 Mutation

211 Following crossover, `POPULATION` array elements affecting agent actions (columns 8-13) mutate at a fixed
 212 probability of `ga_mutation`. For each array element, a random uniform number $u \in [0, 1]$ is sampled. If u is
 213 greater than $1 - (0.5 * \text{ga_mutation})$, then the value of the array element is increased by 1. If u is less
 214 than $0.5 * \text{ga_mutation}$, then the value of the array element is decreased by 1; when this decrease results
 215 in a negative value, the mutated value is multiplied by -1 to be positive.

216 Cost constraint

217 Variation in manager or user actions generated by crossover and mutation might result in strategies that
 218 exceed `manager_budget` or `user_budget`, respectively. Left unchecked, this over-budgeting could lead to
 219 unacceptably high fitness strategies, so strategies that are over budget following crossover and mutation
 220 need to be brought back within budgetary constraints. To do this, the genetic algorithm first checks to see if
 221 an individual in `POPULATION` is over budget. If so, then an action is randomly selected and removed, and

222 budget use is reassessed; this random removal of an action and subsequent budget reassessment continues
223 until the individual does not exceed their budget.

224 **Fitness evaluation**

225 Once all individuals in POPULATION are within budget, the fitness of each individual is assessed. Fitness
226 assessment works differently for managers versus users because managers need to consider the consequences of
227 their decisions on user actions, and how those actions will affect resource abundance. In contrast, user actions
228 need to consider the consequences of their decisions on resource abundance or landscape output. Individual
229 fitness is defined by a real number that increases with the degree to which an individual's actions are predicted
230 to increase entities of positive utility and decrease entities of negative utility (recall that managers and users
231 assign resources or landscape output a utility value). Details for how fitness is calculated are provided below.

232 **Tournament selection**

233 After each individual in POPULATION is assigned a fitness, a tournament is used to select individuals. Tournamen
234 tournament selection is an especially flexible, non-parametric method that samples a subset of individuals from
235 the total population and chooses the fittest of the subset for replacement (Hamblin, 2013). In GMSE,
236 tournament selection proceeds by randomly sampling `ga_sampleK` individuals from the total POPULATION
237 with replacement. The fitnesses of the subset of `ga_sampleK` individuals are compared, and the `ga_chooseK`
238 individuals of highest fitness are retained (if `ga_sampleK` \geq `ga_chooseK`, then all `ga_sampleK` are chosen, but
239 this will prevent adaptive evolution and is therefore not recommended). Tournaments selecting `ga_chooseK`
240 individuals from random subsets of size `ga_sampleK` continue until a total of `ga_popsize` individuals are
241 retained.

242 **Replacement and termination**

243 Once a new set of `ga_popsize` individuals is retained through tournament selection, these individuals replace
244 the previous POPULATION array. The genetic algorithm terminates if and only if a minimum number of
245 iterations has passed (`ga_mingen`) and a convergence criteria (`converge_crit`) is satisfied. The convergence
246 criteria checks the difference between the mean fitness of individuals in the new iteration versus the previous
247 iteration; if this difference is greater than `converge_crit`, then termination does not occur (this prevents
248 termination from occurring while fitness is still increasing, though it is usually fine to use the default GMSE
249 `converge_crit` = 0.1 and `ga_mingen` = 40, which nearly always terminates the genetic algorithm after 40
250 iterations having identified adaptive manager or user strategies). Due to the way in which fitness is calculated
251 (see below), in practice, `converge_crit` currently applies only to users. If termination conditions are not
252 satisfied, then the POPULATION of individuals begins a new iteration of crossover, mutation, cost constraint,
253 fitness evaluation, and tournament selection (Figure 1).

254 **Detailed explanation of manager and user fitness functions**

255 Here we explain how the fitnesses of candidate manager and user strategies in a POPULATION array (see
256 above) are calculated. We emphasise that the fitness functions used in GMSE v0.4.0.7 are intended to be
257 heuristic tools for identifying reasonable manager and user behaviours. In practice, our fitness functions
258 identify behaviours that are well-aligned with manager and user interests for harvesting or crop yield, but
259 they are not intended to identify *optimal* decisions. This practical, metaheuristic approach is consistent with
260 the objectives of management strategy evaluation (Bunnefeld et al., 2011), and is well-suited for the use
261 of genetic algorithms (Hamblin, 2013). Luke (2009) describes the metaheuristic approach more generally
262 (original emphasis retained):

263 Metaheuristics are applied to *I know it when I see it* problems. They're algorithms used to find
 264 answers to problems when you have very little to help you: you don't know beforehand what
 265 the optimal solution looks like, you don't know how to go about finding it in a principled way,
 266 you have very little heuristic information to go on, and brute-force search is out of the question
 267 because the space is too large. *But* if you're given a candidate solution to your problem, you *can*
 268 test it and assess how good it is. That is, you know a good one when you see it.

269 Given the complexity of adaptive management and socio-ecological interactions, the above conditions for
 270 applying the metaheuristic approach are clearly satisfied for manager and user decisions. With this in mind,
 271 we now explain the details of manager and user fitness functions; that is, how GMSE assesses whether or not
 272 a strategy is a good one.

273 Fitness function for managers

274 Individual fitness as calculated for managers (F_i^m) is affected by a manager's utility for resources and
 275 the projected change in resource abundance caused by the individual's policy (i.e., the contents of their
 276 POPULATION layer, specifically row 3; here again we use 'individual' to refer to one of `ga_popsizes` discrete
 277 strategies in POPULATION, which may be selected and reproduce within the genetic algorithm). Manager utility
 278 for a resource (U_{res}^m) is defined as the difference between `manage_target` and the estimation of population
 279 abundance as produced by the GMSE observation model (see "[Key data structures used](#)" above, and [SI7](#)
 280 for more information). Manager utility can therefore change in each GMSE time step as estimated resource
 281 abundance changes; when the estimated resource abundance is greater than `manage_target`, U_{res}^m is negative,
 282 and when the estimated resource abundance is less than `manage_target`, U_{res}^m is positive. To get the fitness
 283 of individuals, first the change in resource abundance predicted by the individual's policy (ΔA_i) is calculated,
 284 then the squared difference between ΔA_i and U_{res}^m is calculated to obtain a utility deviation (D_i) for the
 285 individual i ,

$$D_i = (\Delta A_i - U_{res}^m)^2.$$

286 The value of D_i increases as ΔA_i gets further from U_{res}^m ; i.e, D_i is high when i sets a policy that is not
 287 predicted to get closer to the `manage_target` abundance. Fitness is defined by first finding the maximum D_i
 288 value among all `ga_popsizes` individuals (D_{max}), then subtracting D_i from this value for each individual,

$$F_i^m = D_{max} - D_i.$$

289 We have explained how U_{res}^m is calculated in the [above section on key data structures](#). We now explain in
 290 more detail how individuals in the genetic algorithm calculate how their actions will affect ΔA_i .

291 To predict change in resource abundance as a consequence of policy, an individual first needs to know the
 292 total number of actions of all types j (e.g., scaring, culling, etc.) performed by users in the previous time
 293 step ($X_{\bullet,j}$; note that this value includes the increment `manage_caution`, with a default of `manage_caution`
 294 = 1, to ensure that managers do not naïvely assume that users will not perform an action just because they
 295 did not perform it in the previous time step), and the cost of performing each action ($C_{\bullet,j}$). This information
 296 is collected from ACTION and COST arrays. The individual i then needs to predict how their policy (i.e., the
 297 costs that they set for users to perform an action) will affect the new total number of each action j performed
 298 ($X_{i,j}$). To do this, the individual assumes that total user actions performed under their policy will change in
 299 proportion to that of the old policy, while also recognising that users have a maximum above which higher
 300 costs set by the manager will have no effect. Interested readers might wish to examine the short `new_act`
 301 function, which is summarised mathematically below; this function is called by the `policy_to_counts`
 302 function in the [genetic algorithm source file](#).

303 The manager first calculates how much total budget, as summed over all users, was devoted to an action by
 304 multiplying the old per action cost $C_{\bullet,j}$ by the total number of actions performed, $X_{\bullet,j}$. The manager then
 305 divides this by the new cost $C_{i,j}$ per action to calculate the new predicted number of actions,

$$X_{i,j} = \frac{X_{\bullet,j} \times C_{\bullet,j}}{C_{i,j}}.$$

306 Note again that if $C_{i,j} = C_{\bullet,j}$, then the total number of new predicted actions j will remain unchanged. If
 307 $C_{i,j} > C_{\bullet,j}$, then the total number of new actions will decrease, and if $C_{i,j} < C_{\bullet,j}$, then the total number of
 308 new actions will increase.

309 The predicted consequences of $X_{i,j}$ for resource abundance differ for each possible action. For each action, no
 310 consequence is predicted if the policy is not allowed by a simulation of GMSE (e.g., `culling = FALSE`). For
 311 allowed actions, the parameter `manager_sense` (σ) modulates predicted consequences for abundance by some
 312 factor; this is useful because not all actions attempted by users will be realised, and a value of $\sigma = 1$ tends to
 313 slightly overestimate how much the actions attempted by users will actually translate to a change in resource
 314 abundance. In practice, the default $\sigma = 0.9$ performs well. Allowed actions are predicted by managers to
 315 have the following effects (again, we emphasise that whether or not these effects are realised will depend later
 316 on the user model, to which the manager – by design – does not have access):

- 317 • `scaring` is assumed to be nonlethal and therefore have no effect on resource number (resources are
 318 moved to a random cell on the landscape, as sampled from a uniform distribution such that movement
 319 to any given cell is equally probable).
- 320 • `culling` decreases resource number by σ .
- 321 • `castration` decreases resource number by $\sigma\lambda$, where λ is the GMSE argument `lambda` that defines the
 322 baseline population growth rate of resources.
- 323 • `feeding` increases resource number by $\sigma\lambda$.
- 324 • `help_offspring` increases resource number by σ .

325 Note that σ is included in all of the predicted actions above as a modulator for how strongly the manager
 326 predicts users will respond to a change in manager policy (e.g., a value of 0 would predict no reaction on the
 327 part of users to a change in policy, while a value of 1 would predict that an action would increase in exact
 328 proportion to its decrease in cost).

329 The above effects cannot be altered directly in `gmse` or `gmse_apply` (though parameter values can of course be
 330 changed using `manager_sense` and `lambda` arguments), but future versions of GMSE might include different
 331 predicted effects to increase precision or allow for multiple resource types or different actions. The summation
 332 of $X_{i,j}$ for all actions defines the predicted change in resource abundance caused by the policy of an individual
 333 i , ΔA_i .

334 Fitness function for users

335 The previous section described the fitness function applied when individual’s fitness was evaluated for
 336 managers; here we explain a separate fitness function that is applied when individuals are instead evaluated
 337 for users. Individual fitness as calculated for users (F_i^u) is affected by a user’s utility for resources (U_{res}^u) and
 338 landscape output (U_{land}^u), and the predicted change in each caused by the user’s actions (ΔA_i and ΔL_i for
 339 predicted change in resource abundance and summed values of the landscape cells owned by i , respectively).
 340 Individual fitness is defined for users below,

$$F_i^u = \Delta A_i U_{res}^u + \Delta L_i U_{land}^u.$$

341 Note that F_i^u increases when ΔA_i and ΔL_i are of the same sign as U_{res}^u and U_{land}^u , respectively. Further,
 342 in GMSE v0.4.0.7, only one term of the equation is nonzero. When `land_ownership = FALSE` (default,
 343 modelling users that harvest resources), $U_{res}^u = -1$ and $U_{land}^u = 0$, and when `land_ownership = TRUE`,
 344 $U_{res}^u = 0$ and $U_{land}^u = 100$ (modelling farmers trying to increase crop yield). Hence users only have a single
 345 objective of either decreasing resource abundance or increasing landscape output, though landscape output
 346 might be increased indirectly by decreasing resource abundance if `resource_consume` is greater than zero.

347 User actions are predicted to affect resources in the following way:

- 348 • `scaring` decreases resource number by 1.
- 349 • `culling` decreases resource number by 1.
- 350 • `castration` decreases resource number by λ .
- 351 • `feeding` increases resource number by λ .
- 352 • `help_offspring` increases resource number by 1.

353 The number of each action performed is multiplied by its effect, and the sum of all these products is the
354 predicted ΔA_i ,

$$\Delta A_i = (\lambda)Feeds + Helps - Scares - Culls - (\lambda)Castrations.$$

355 There are only two possible actions that users can take to directly affect landscape output, tending crops
356 (`tend_crops`) and killing crops (`kill_crops`). The increase in landscape output is modulated by the
357 parameter `tend_crop_yld` (ϕ). User actions are therefore predicted to have the following effects for one
358 landscape cell:

- 359 • `tend_crops` will increase landscape output by ϕ .
- 360 • `kill_crops` will decrease landscape output by 1 (since the output of a cell is 1 by default, this action
361 removes all output on a landscape cell).

362 Actions on resources can also have indirect effects on ΔL_i when resources consume output on the landscape;
363 we define the value `res_consume` as r . The predicted ΔL_i is then,

$$\Delta L_i = (\phi)Tends - Kills - r\Delta A_i.$$

364 That is, the change in landscape output equals the increase in output from tending crops, minus the number
365 of crops destroyed, minus the change in resource abundance times the effect that resource abundance has on
366 landscape output (note that if user actions decrease resource abundance, then this last term will be positive,
367 increasing landscape output).

368 Choosing genetic algorithm parameter values

369 Options for adjusting genetic algorithm parameter values in `gmse` and `gmse_apply` are shown below.

GMSE argument	Default	Description
<code>ga_popsize</code>	100	The number of individuals in the population temporarily simulated during a single run of the genetic algorithm.
<code>ga_mingen</code>	40	The minimum number of iterations that a genetic algorithm will run before settling on an agent's strategy.
<code>ga_seedrep</code>	20	The number of individuals in the population to be initialised with the current agent's strategy (e.g., from a previous time step in the broader GMSE simulation), as opposed to being initialised with random strategies.
<code>ga_sampleK</code>	20	For the tournament step of the genetic algorithm, how many strategies are selected at random from the larger population (with replacement) to be included in the tournament.
<code>ga_chooseK</code>	2	For the tournament step of the genetic algorithm, how many strategies are selected as winners of the tournament, to be included in the next iteration.
<code>ga_mutation</code>	0.1	The mutation rate of any action in an agent's strategy

GMSE argument	Default	Description
<code>ga_crossover</code>	0.1	The crossover rate of any action in an agent’s strategy; crossover events occur with a different randomly selected strategy in the population.
<code>ga_converge_crit</code>	0.1	The percent increase in strategy fitness from one iteration to the next below which the convergence criteria is satisfied. Iterations will continue as long as fitness increase is above this convergence criteria.
<code>group_think</code>	FALSE	Whether or not all users (i.e., not including the manager) have identical strategies. If TRUE, then one genetic algorithm will be run and applied to all users.

Given the heuristic goals of the genetic algorithm to mimic the goal-oriented behaviour of agents, default parameters are typically sufficient for agent decision making. Key parameters can be adjusted if more precision in decision making is desired, but these adjustments will come at a cost of simulation efficiency. For example, increasing `ga_popsize` or `ga_mingen`, or decreasing `ga_converge_crit`, might fine tune strategies more effectively, but this will cause the genetic algorithm to take longer every time that it is run, ultimately slowing down GMSE simulations. Alternatively, setting `group_think = TRUE` will greatly speed up GMSE simulations when many users are being simulated, but this comes at the cost of among-user variation in decision making. Overall, we recommend first using default values in the genetic algorithm before exploring how other parameter value options affect simulation dynamics; for a more general discussion about selecting parameter values in genetic algorithms, see Hamblin (2013).

Future development of fitness functions

The fitness functions defined above are useful heuristics for simulating manager and user decision-making in a way that produces realistic, *I know it when I see it*, strategies. Future versions of GMSE might improve upon these heuristics to generate more accurate or more realistic models of human decision making. Such improvements could incorporate additional information such as memory of actions from multiple past time steps, or a continually updated estimate for how actions are predicted to affect resource abundance or landscape output in a simulation (e.g., through a dynamic `manager_sense`). Alternatively, future improvements could usefully incorporate knowledge of human decision making collected from empirical observation of human behaviour during conservation conflicts. While such possibilities could be useful for future GMSE modelling, repeated simulations demonstrate the ability of the current GMSE genetic algorithm to find adaptive strategies for managers attempting to keep resources at target abundance, and users attempting to maximise their harvests or crop yields. It is therefore useful as a tool for modelling manager and user decisions in a generalised management strategy evaluation framework.

References

- Adami, C., Schossau, J., and Hintze, A. (2016). Evolutionary game theory using agent-based methods. *Physics of Life Reviews*, 19:1–26.
- Balmann, A. and Happe, K. (2000). Applying parallel genetic algorithms to economic problems: The case of agricultural land markets. In *IIFET Conference “Microbehavior and Macroresults”. Proceedings.*, Corvallis, Oregon, USA.
- Bunnefeld, N., Hoshino, E., and Milner-Gulland, E. J. (2011). Management strategy evaluation: A powerful tool for conservation? *Trends in Ecology and Evolution*, 26(9):441–447.
- Hamblin, S. (2013). On the practical usage of genetic algorithms in ecology and evolution. *Methods in Ecology and Evolution*, 4(2):184–194.

- 403 Kark, S., Tulloch, A., Gordon, A., Mazor, T., Bunnefeld, N., and Levin, N. (2015). Cross-boundary
404 collaboration: Key to the conservation puzzle. *Current Opinion in Environmental Sustainability*, 12:12–24.
- 405 Lee, C. S. (2012). Multi-objective game-theory models for conflict analysis in reservoir watershed management.
406 *Chemosphere*, 87(6):608–613.
- 407 Luke, S. (2009). *Essentials of Metaheuristics*. Lulu. Available for free at
408 <http://cs.gmu.edu/~sean/book/metaheuristics/>.
- 409 Redpath, S. M., Keane, A., Andrén, H., Baynham-Herd, Z., Bunnefeld, N., Duthie, A. B., Frank, J., Garcia,
410 C. A., Månsson, J., Nilsson, L., Pollard, C. R. J., Rakotonarivo, O. S., Salk, C. F., and Travers, H. (2018).
411 Games as Tools to Address Conservation Conflicts. *Trends in Ecology and Evolution*, 33(6):415–426.
- 412 Tilman, A. R., Watson, J. R., and Levin, S. (2016). Maintaining cooperation in social-ecological systems:.
413 *Theoretical Ecology*.
- 414 Tu, M. T., Wolff, E., and Lamersdorf, W. (2000). Genetic algorithms for automated negotiations: a FSM-
415 based application approach. *Proceedings 11th International Workshop on Database and Expert Systems*
416 *Applications*, pages 1029–1033.