GMSE: an R package for generalised management strategy evaluation

Supporting Information 1

A. Bradley Duthie, Jeremy J. Cusack, Isabel L. Jones, Erlend B. Nilsen, Rocío Pozo, O. Sarobidy Rakotonarivo, Bram Van Moorter, and Nils Bunnefeld

2017-11-08

Extended introduction to the genetic algorithm applied in GMSE

A genetic algorithm is called in the predefined GMSE manager and user models to simulate human decision making. As of GMSE version 0.3.1.9, 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), 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 follow identically. Each run of the genetic algorithm mimics the evolution by natural selection of a population of potential manager or user strategies over multiple generations, with the highest fitness strategy in the terminal generation 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 'generation' 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 v0.3.1.9 (future versions of GMSE might expand upon this framework). 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`). 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 layer of `ACTION` corresponds to a unique agent, the first of which is the manager; additional layers correspond to users. Below shows an `ACTION` array for a GMSE model with one manager and two users.

```
## , , Manager_Actions
##
##           Act Type_1 Type_2 Type_3     Util. U_land U_loc. Scare Cull
## Resource   -2      1      0      0 1000.0000      0      0     0    0
## Landscape  -1      1      0      0    0.0000      0      0     0    0
## Res_cost    1      1      0      0  229.0249      0      0    10  110
## U1_cost     2      1      0      0    0.0000      0      0     0    0
## U2_cost     3      1      0      0    0.0000      0      0     0    0
##           Castrate Feed Help_off None
## Resource         0    0        0    0
## Landscape        0    0        0    0
```

```
## Res_cost        10   10       10   10
## U1_cost          0    0        0    0
## U2_cost          0    0        0    0
##
## , , User_1_Actions
##
##           Act Type_1 Type_2 Type_3 Util. U_land U_loc. Scare Cull Castrate
## Resource   -2      1      0      0    -1      0      0     0    9        0
## Landscape  -1      1      0      0     0      0      0     0    0        0
## Res_cost    1      1      0      0     0      0      0     0    0        0
## U1_cost     2      1      0      0     0      0      0     0    0        0
## U2_cost     3      1      0      0     0      0      0     0    0        0
##           Feed Help_off None
## Resource     0        0    0
## Landscape    0        0    1
## Res_cost     0        0    0
## U1_cost      0        0    0
## U2_cost      0        0    0
##
## , , User_2_Actions
##
##           Act Type_1 Type_2 Type_3 Util. U_land U_loc. Scare Cull Castrate
## Resource   -2      1      0      0    -1      0      0     0    9        0
## Landscape  -1      1      0      0     0      0      0     0    0        0
## Res_cost    1      1      0      0     0      0      0     0    0        0
## U1_cost     2      1      0      0     0      0      0     0    0        0
## U2_cost     3      1      0      0     0      0      0     0    0        0
##           Feed Help_off None
## Resource     0        0    0
## Landscape    0        0    1
## Res_cost     0        0    0
## U1_cost      0        0    0
## U2_cost      0        0    0
```

The above array holds all of the information on manager and user actions. The first seven columns contain information about which entities are affected, and how they are affected. The first column `Act` identifies the type of action being performed; a value of -2 defines a direct action to a resource (e.g., culling of the resource), and a value of -1 defines direct action to a landscape (e.g., increasing yield). Positive values are currently only meaningful for `Manager_Actions`, where a value of 1 defines an action setting a uniform cost of users' direct actions on resources (i.e., costs where `Act = -2` for `User_1_Actions` and `User_2_Actions`). All other values for `Act` are meaningless in GMSE 0.3.1.9, but might be expanded upon in future versions to allow for modification of specific user costs enacted by managers (i.e., managers having different policies for different users) or other users (e.g., users increasing the costs of other users' actions due to conflict or cooperation). Similarly, columns 2-4 refer to resource or landscape types, but only `Type_1 = 1`, `Type_2 = 0`, and `Type_3 = 0` are allowed in GMSE v0.3.1.9 (i.e., only one type of resource is permitted). Future versions might allow for different resource types (e.g., `Type_1` might be used to designate species, and `Type_2` and `Type_3` could designate stage or sex). For the rest of this supporting information, we will therefore focus only on rows 1-3 of `ACTION`. Column 5 `Util.` of `ACTION` defines the utility associated with the resource (where `Act = -2`) or landscape (where `Act = -1`). For managers, the target resource abundance set with the GMSE argument `manage_target` is found in row 1 (1000 in `ACTION` above); for users, the value in row 1 identifies whether resources are preferred to increase (if positive) or decrease (if negative). Values of column 5 in row 2 similarly identify whether landscape cell output is preferred by users to increase or decrease (managers do not currently have preferences for landscape output). Of special note is row 3 for `Manager_Actions`, which defines the manager's utility of resources; that is, the adjustment to resource abundance that the manager

will attempt to make based on the `manage_target` and the estimated abundance produced by the observation model (in the case of the above, resource abundance is estimated at ca 770.98, so the manager will set policy in attempt to change the population size by ca 229.02 resources). Column 6 `U_land` defines whether or not the utility attached to the resource or landscape output depends on it being on a landscape cell that is owned by the acting user. Related, column 7 `U_loc.` defines whether or not actions can be performed only on a landscape cell that is owned by the acting user. Hence values of columns 6 and 7 are binary, and affected by the `land_ownership` argument in `gmse`. Finally, columns 8-13 correspond to specific actions, either direct (where `Act < 0`) or indirect by setting policy (for row 3 of `Manager_Actions` where `Act = 1`). The last column 13 `None` corresponds with no actions. See GMSE documentation for details about the effects of each action.

Constraints on the values that elements in the `ACTION` array can take are defined by a `COST` array (also returned as `manager_array` by `gmse_apply`) of dimensions identical to `ACTION`. Elements of `COST` define how many units from the `manager_budget` or `user_budget` are needed to perform a single action; a `minimum_cost` for actions is defined as an argument in GMSE (10 by default). All values in `COST` columns 1-7 are set to 10001, one higher than the highest possible `manager_budget` or `user_budget`, so neither can affect resource types or utilities. Columns 8-13 are also set to 10001, except where actions are allowed. Below shows the `COST` array that corresponds to the above `ACTION` array.

```
## , , Manager_Actions
##
##             Act Type_1 Type_2 Type_3 Util. U_land U_loc. Scare  Cull
## Resource  10001  10001  10001  10001 10001  10001  10001 10001 10001
## Landscape 10001  10001  10001  10001 10001  10001  10001 10001 10001
## Res_cost  10001  10001  10001  10001 10001  10001  10001 10001    10
## U1_cost   10001  10001  10001  10001 10001  10001  10001 10001 10001
## U2_cost   10001  10001  10001  10001 10001  10001  10001 10001 10001
##           Castrate  Feed Help_off  None
## Resource     10001 10001    10001    10
## Landscape    10001 10001    10001    10
## Res_cost     10001 10001    10001    10
## U1_cost      10001 10001    10001 10001
## U2_cost      10001 10001    10001 10001
##
## , , User_1_Actions
##
##             Act Type_1 Type_2 Type_3 Util. U_land U_loc. Scare  Cull
## Resource  10001  10001  10001  10001 10001  10001  10001 10001   110
## Landscape 10001  10001  10001  10001 10001  10001  10001 10001 10001
## Res_cost  10001  10001  10001  10001 10001  10001  10001 10001 10001
## U1_cost   10001  10001  10001  10001 10001  10001  10001 10001 10001
## U2_cost   10001  10001  10001  10001 10001  10001  10001 10001 10001
##           Castrate  Feed Help_off  None
## Resource     10001 10001    10001    10
## Landscape    10001 10001    10001    10
## Res_cost     10001 10001    10001 10001
## U1_cost      10001 10001    10001 10001
## U2_cost      10001 10001    10001 10001
##
## , , User_2_Actions
##
##             Act Type_1 Type_2 Type_3 Util. U_land U_loc. Scare  Cull
## Resource  10001  10001  10001  10001 10001  10001  10001 10001   110
## Landscape 10001  10001  10001  10001 10001  10001  10001 10001 10001
## Res_cost  10001  10001  10001  10001 10001  10001  10001 10001 10001
```

```
## U1_cost   10001  10001  10001  10001 10001  10001  10001 10001 10001
## U2_cost   10001  10001  10001  10001 10001  10001  10001 10001 10001
##           Castrate  Feed Help_off  None
## Resource     10001 10001    10001    10
## Landscape    10001 10001    10001    10
## Res_cost     10001 10001    10001 10001
## U1_cost      10001 10001    10001 10001
## U2_cost      10001 10001    10001 10001
```

Note that in default GMSE parameters, `culling = TRUE`, but all other actions are false. Hence the `Cull` column 9 is the only column besides column 13 `None` in which cost is less than 10001. Manager's actions in `ACTION` directly affect the cost of users performing one of the five possible actions on resources (columns 8-12). This can be verified in `ACTION` where the manager has set the cost of scaring to 110, and the corresponding `COST` of resource culling (row 1) is 110 for both users. The cost of the manager affecting the cost of user actions is always set to the `minimum_cost`; here the default 10 is used. This `minimum_cost` also defines cost values for `None`, in which the user or manager does nothing, as might occur if the manager wants to permit culling and therefore does not want to invest any of their `manager_budget` to increasing the cost of culling. Both `ACTION` and `COST` are updated in each time step unless `manage_freq > 1`, in which case `COST` and `Manager_Actions` in `ACTION` are updated at the frequency defined.

General overview of key aspects of the genetic algorithm

The genetic algorithm updates a single layer of the `ACTION` array, which defines to the decisions of a single agent (either the manager or a user). The corresponding layer of the `COST` array remains unchanged, and serves only to ensure that `ACTION` values do not exceed `manager_budget` or `user_budget` for managers and users, respectively. The genetic algorithm proceeds by first initialising a large population of new `ACTION` layers. In each generation, these layers crossover and mutate, generating variation in agent decisions; costs constrain this variation from exceeding a maximum budget, then the fitness of each layer is evaluated based on how much it increases the agent's utility. A tournament is used to select high fitness layers, and these selected layers become the new generation of layers; generations continue until a minimum number of generations (`ga_mingen`) have passed and a convergence criteria is satisfied such that the increase in mean fitness from the previous generation is below the threshold `converge_crit` (Figure 1).

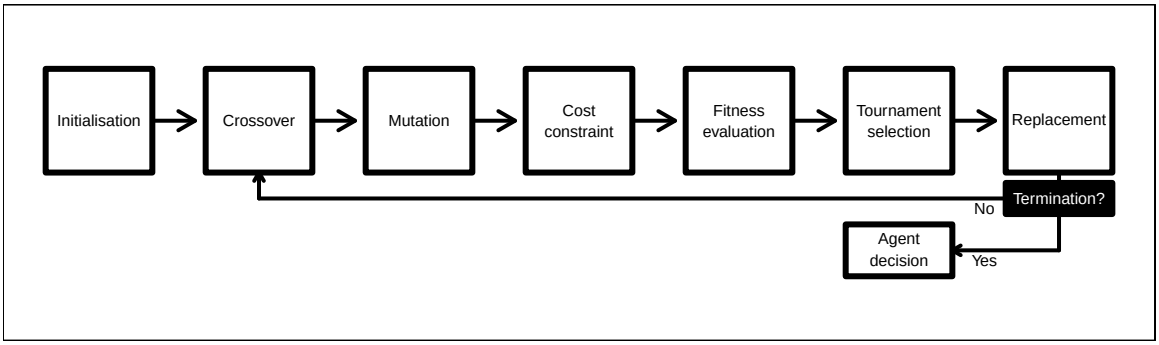

Figure 1: Conceptual overview of the GMSE genetic algorithm

Initialisation

At the start of each genetic algorithm, a population of size `ga_popsize` is initialised (hereafter the `POPULATION` array). This population is held in a 3D array of `ga_popsize` layers. Each layer includes an identical number of rows and columns as in `ACTION`, and one layer defines a single 'individual' in the population. The first seven columns of `ACTION` are replicated exactly for all individuals, and remain unchanged throughout the genetic

algorithm thereby preserving the information about which entities are affected by actions in a given row. The remaining columns are either also replicated exactly as in `ACTION` (i.e., initialised to be the same decisions as in a previous time step), or randomly seeded with values given the constraints of `manager_budget` or `user_budget` (i.e., initialised to random decision making). The number of exact replicates initialised is set using `ga_seedrep` (if `ga_seedrep` $\geq$ `ga_popsize`, then all individuals are seeded as replicates). After the `POPULATION` of `ga_popsize` individuals is initialised, a loop simulating the adaptive evolution of `POPULATION` in non-overlapping generations begins.

Crossover

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

Mutation

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

Cost constraint

Variation in manager or user actions generated by crossover and mutation might result in strategies that exceed `manager_budget` or `user_budget`, respectively. Left unchecked, this over-budgeting could lead to unnacceptably high fitness strategies, so strategies that are over budget following crossover and mutation need to be brought back within budgetary constraints. To do this, the genetic algorithm first checks to see if an individual in `POPULATION` is over budget. If so, then an action is randomly selected and removed, and budget use is reassessed; this random removal of an action and subsequent budget reassessment continues until the individual does not exceed their budget.

Fitness evaluation

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

Tournament selection

After each individual in `POPULATION` is assigned a fitness, a tournament is used to select individuals of higher fitness. Tournament selection is an especially flexible, non-parametric method that samples a subset of individuals from the total population and chooses the fittest of the subset for replacement (Hamblin, 2013). In GMSE, tournament selection proceeds by randomly sampling `ga_sampleK` individuals from the total `POPULATION` with replacement. The fitnesses of the subset of `ga_sampleK` individuals are compared, and the

`ga_chooseK` individuals of highest fitness are retained (if `ga_sampleK` $\geq$ `ga_chooseK`, then all `ga_sampleK` are chosen, but this is not recommended). Tournaments selecting `ga_chooseK` individuals from random subsets of size `ga_sampleK` continue until a total of `ga_popsize` individuals are retained.

Replacement and termination

Once a new set of `ga_popsize` individuals is retained through tournament selection, these individuals replace the previous `POPULATION` array. The genetic algorithm terminates if and only if a minimum number of generations has passed (`ga_mingen`) and a convergence criteria (`converge_crit`) is satisfied. The convergence criteria checks the difference between the mean fitness of individuals in the new generation versus the previous generation; if this difference is less than `converge_crit`, then termination does not occur (it is usually fine to use the default GMSE `converge_crit = 100` and `ga_mingen = 40`, which nearly always terminates the genetic algorithm after 40 generations having identified adaptive manager or user strategies). If termination conditions are not satisfied, then the `POPULATION` of individuals begins a new generation of crossover, mutation, cost constraint, fitness evaluation, and tournament selection (Figure 1).

Detailed explanation of manager and user fitness functions

Here we explain how candidate manager and user fitness strategies in a `POPULATION` array (see above) are calculated. We emphasise that the fitness functions used in GMSE v0.3.1.9 are intended to be heuristic tools for identifying reasonable manager and user behaviours. In practice, our fitness functions identify behaviours that are well-aligned with manager and user interests for harvesting or crop yield, but they are not intended to identify *optimal* decisions. This practical, metaheuristic approach is consistent with the objectives of Management Strategy Evaluation (Bunnefeld et al., 2011), and is well-suited for the use of genetic algorithms (Hamblin, 2013). Luke (2013) describes the metaheuristic approach more generally (original emphasis retained):

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

The above conditions for applying the metaheuristic approach are clearly satisfied for manager and user decisions, given the complexity of adaptive management and socio-ecological interactions.

Fitness function for managers

Individual fitness as calculated for managers (F_i^m) is affected by a manager's utility for resources and the projected change in resource abundance caused by the individual's policy (i.e., the contents of their `POPULATION` layer, specifically row 3). Manager utility for a resource (U_{res}^m) is defined as the difference between `manage_target` and the estimation of population abundance as produced by the GMSE observation model (see "Key data structures used" above). Manager utility can therefore change in each GMSE time step as resource abundance (and its estimate from the observation model) changes; when the estimated resource abundance is greater than `manage_target`, U_{res}^m is negative, and when the estimated resource abundance is less than `manage_target`, U^m is positive. To get individual fitness, first the change in resource abundance predicted by the individual's policy (ΔA_i) is calculated, and the squared difference between ΔA_i and U_{res}^m is calculated to obtain a utility deviation for the individual i,

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

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

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

We have explained how U^m_{res} is calculated in the above section on key data structures. We now explain in more detail how individuals in the genetic algorithm calculate their actions will affect ΔA_i. Below explains calculations for a single individual i in the genetic algorithm, so we will drop the use of subscripts i to refer to individuals; instead we use subscripts j to refer to actions (e.g., scaring, culling, etc.).

To predict change resource abundance as a consequence of policy, an individual first needs to know the total number of actions of all types j performed by users in the previous time step (X^{old}_j), and the cost of performing each action (C^j_{old}). This information is collected from `ACTION` and `COST` arrays. The individual then needs to predict how their policy (i.e., the costs that they set users to perform an action) will affect the new total number of each action performed (X^{new}_j). To do this, the individual assumes that total user actions performed under their policy will change in proportion to that of the old policy. The predicted total number of a particular action j performed is thereby calculated as,

$$X^{new}_j = (X^{old}_j + 1)\frac{C^{old}_j}{C^{new}_j}.$$

The variable C^j_{new} defines the new cost set by the individual for action j. A value of 1 is added to (X^{old}_j) to model some degree of caution by the manager (this can be changed from the default 1 using `manage_caution`), especially so that managers do not naïvely assume that users will not perform an action just because they did not perform it in the previous time step. Otherwise, if $X^{old}_j = 0$, then the manager would always assume that a change in the cost of an action would have no affect on the number of times the action was performed by users; a value of 1 assumes that at least one user will perform the action in the new time step.

The predicted consequences of X^{new}_j for resource abundance differ for each possible action. For each action, no consequence is predicted if the policy is not allowed by a simulation of GMSE (e.g., `scaring = FALSE`).

Fitness function for users

Thanks for the clarification regarding the equation. I'll try to answer as best as I can – apologies if this has been unclear. At the broadest scale, the equation for user fitness would be on L367 in the strategy_fitness function (https://github.com/bradduthie/gmse/blob/master/src/game.c#L376). Here's what's going on: Users are predicting how their actions will change the quantities of things in the model (either resources or landscape output), and these changes are individually multiplied by the users' utilities for that thing. The change multiplied by utility for each thing is summed across all things to get a value for fitness. Note that positive change times positive utility, and negative change times negative utility, will increase fitness (i.e., increasing the thing users want more of and decreasing the things they want less of). Hence, an equation describing user fitness would be the below,

$$F_{user} = \sum_{i=1}^{N} \Delta A_i \times U_i$$

.

Where F_{user} is user fitness, N is the total number of things that might be of interest (at the moment $N = 2$ in GMSE, one resource and, potentially, one landscape value), ΔA_i is the change in the abundance of thing i, and U_i is the utility of thing i from the perspective of the user (apologies for the LaTeX code – attached a PNG of the conversion). I want to stress though that I would not consider this equation to be central to the

GMSE framework – if someone else has a better approach for defining fitness, or defining any of the terms listed above, or wants to expand upon it to include new things, then that would be awesome! The above just works well as a heuristic tool to get users to act in such a way as to maximise their interests in harvesting or getting more crop yield (as is my intent), but it's not based on first principles and I don't claim it to be particularly special.

The values of ΔA_i are calculated for resources and the landscape in the functions res_to_counts and land_to_counts, respectively (and U_i is specified a priori in the model depending on other parameters – namely land_ownership). Again, a bit of heuristic is needed here because there cannot be any perfect way of exactly predicting how a users actions will increase or decrease resources – there are too many complex factors (e.g., behaviour of other stakeholders, demographic stochasticity, movement of resources on the landscape, and interactions between resources and the landscape). Even if we could include all of these things somehow, it would be a bit unrealistic in that real stakeholders would never have this much information. The predicted direct effect of actions on resources is shown in lines 268-272 (https://github.com/bradduthie/gmse/blob/master/src/game.c#L268), and the array 'jaco' (a sort of Jacobian matrix) accounts for interactions between landscape and resources on line 286. Something similar happens in the land_to_counts function. The manager's genetic algorithm works in a similar way (the above equation applies), but with the need to dynamically update utility values based on current resource abundance, and to account for the predicted actions of users in finding ΔA_i.

References

Bunnefeld, N., Hoshino, E., and Milner-Gulland, E. J. (2011). Management strategy evaluation: A powerful tool for conservation? *Trends in Ecology and Evolution*, 26:441–447.

Hamblin, S. (2013). On the practical usage of genetic algorithms in ecology and evolution. *Methods in Ecology and Evolution*, 4:184–194.

Luke, S. (2013). *Essentials of Metaheuristics*. Lulu, second edition, available at https://cs.gmu.edu/~sean/book/metaheuristics/Essentials.pdf.