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

2017-11-10

# 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 copy). 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, 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). 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 0 160.9977 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 0  
## 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). For the rest of this supporting information, we will therefore focus only on rows 1-3 of ACTION.

Columns 2-4 refer to resource or landscape types, but only Type\_1 = 1, Type\_2 = 0, and Type\_3 = 0 are allowed in predefined GMSE v0.3.1.9 manager and user sub-models (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). 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 *current* manager's utility for resources; that is, the adjustment to resource abundance that the manager will attempt to make based on the manage\_target and the most recent estimate of resource abundance produced by the observation model (in the case of the above, resource abundance is estimated at ca 839, so the manager will set policy in attempt to change the population size by ca 161 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 and gmse\_apply. 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](https://cran.r-project.org/web/packages/GMSE/GMSE.pdf) for details about the effects of each action.

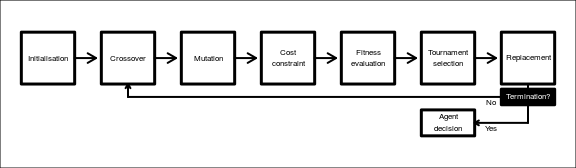
Constraints on the values that elements in the ACTION array can take are defined by a COST array (also returned as manager\_array by gmse\_apply) of dimensions identical to ACTION. Elements of COST define how many units from the manager\_budget or user\_budget are needed to perform a single action; a 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 managers nor users 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\_Costs  
##   
## 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\_Costs  
##   
## 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\_Costs  
##   
## 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 set to 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 culling to 110 (row 3), and the corresponding COST of resource culling is 110 for both users (row 1). 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 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 (but temporary) population of new ACTION layers. In each generation, these layers crossover and mutate, generating variation in potential agent decisions; costs constrain this variation from exceeding a maximum budget, then the fitness of each layer is evaluated based on how the layer is predicted to affect resources or landscape output to which the agent has assigned some 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 S1-1).



**Figure S1-1:** 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 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 (see Figure S1-1 above).

## 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 is sampled. If is greater than 1 - (0.5 \* ga\_mutation), then the value of the array element is increased by 1. If 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. Individual fitness is defined by a real number that increases with the degree to which an individual's actions are predicted to increase entities of positive utility and decrease entities of negative 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. 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 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 greater than converge\_crit, then termination does not occur (this prevents termination from occuring while fitness is still increasing, though 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). Due to the way in which fitness is calculated (see below), in practice, converge\_crit currently applies only to users. 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 S1-1).

# Detailed explanation of manager and user fitness functions

Here we explain how the fitnesses of candidate manager and user 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, Hoshino, and Milner-Gulland 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.

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

## Fitness function for managers

Individual fitness as calculated for managers () 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 () 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](#Key_data_structures_used)" above). Manager utility can therefore change in each GMSE time step as estimated resource abundance changes; when the estimated resource abundance is greater than manage\_target, is negative, and when the estimated resource abundance is less than manage\_target, is positive. To get individual fitness, first the change in resource abundance predicted by the individual's policy () is calculated, then the squared difference between and is calculated to obtain a utility deviation () for the individual ,

The value of increases as gets further from ; i.e, is high when sets a policy that is not predicted to get closer to the manage\_target abundance. Fitness is defined by first finding the maximum value among all ga\_popsize individuals (), then subtracting from this value for each individual,

We have explained how is calculated in the [above section on key data structures](#Key_data_structures_used). We now explain in more detail how individuals in the genetic algorithm calculate how their actions will affect .

To predict change resource abundance as a consequence of policy, an individual first needs to know the total number of actions of all types (e.g., scaring, culling, etc.) performed by users in the previous time step (), and the cost of performing each action (). 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 for users to perform an action) will affect the new total number of each action performed (). 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 performed under the policy of is thereby calculated as,

The variable defines the new cost set by the individual for action . A value of 1 is added to () 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 , then the manager would always assume that a change in the cost of an action would have no effect 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 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., culling = FALSE). For allowed actions, the parameter manager\_sense () modulates predicted consequences for abundance by some factor; this is useful because not all actions attempted by users will be realised, and a value of tends to greatly overestimate how much the actions attempted by users will actually translate to a change in resource abundance. In practice, the default performs well. Allowed actions are predicted by managers to have the following effects:

* scaring is assumed to be nonlethal and therefore have no effect on resource number.
* culling decreases resource number by , where is the GMSE argument lambda that defines the baseline population growth rate of resources.
* castration decreases resource number by .
* feeding increases resource number by .
* help\_offspring increases resource number by .

These effects cannot be altered directly in gmse or gmse\_apply (though parameter values can of course be changed using manager\_sense and lambda arguments), but future versions of GMSE might include different predicted effects to increase precision or allow for multiple resource types or different actions. The summation of for all actions defines the predicted change in resource abundance caused by the policy of an individual , .

## Fitness function for users

The previous section described the fitness function applied when individual's fitness was evaluated for managers; here we explain a separate fitness function that is applied when individuals are instead evaluated for users. Individual fitness as calculated for users () is affected by a user's utility for resources () and landscape output (), and the predicted change in each caused by the user's actions ( and for predicted change in resource abundance and summed values of the landscape cells owned by , respectively). Individual fitness is defined for users below,

Note that increases when and are of the same sign as and , respectively. Further, in GMSE v0.3.1.9, only one term of the equation is nonzero. When land\_ownership = FALSE (default, modelling users that harvest resources), and , and when land\_ownership = TRUE, and (modelling farmers trying to increase crop yield). Hence users only have a single objective of either decreasing resource abundance or increasing landscape output, though landscape output might be increased indirectly by decreasing resource abundance if resource\_consume is greater than zero.

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

* scaring decreases resource number by .
* culling decreases resource number by .
* castration decreases resource number by .
* feeding increases resource number by .
* help\_offspring increases resource number by .

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

There are only two possible actions that users can take to directly affect landscape output, tending crops (tend\_crops) and killing crops (kill\_crops). The increase in landscape output is modulated by the parameter tend\_crop\_yld (). User actions are therefore predicted to have the following effects for one landscape cell:

* tend\_crops will increase landscape output by .
* kill\_crops will decrease landscape output by 1 (since the output of a cell is 1 by default, this action removes all output on a landscape cell).

Actions on resources can also have indirect effects on when resources consume output on the landscape; we define the value res\_consume as . The predicted is then,

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

## 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

Bunnefeld, Nils, Eriko Hoshino, and Eleanor J Milner-Gulland. 2011. “Management strategy evaluation: A powerful tool for conservation?” *Trends in Ecology and Evolution* 26: 441–47. doi:[10.1016/j.tree.2011.05.003](https://doi.org/10.1016/j.tree.2011.05.003).

Hamblin, Steven. 2013. “On the practical usage of genetic algorithms in ecology and evolution.” *Methods in Ecology and Evolution* 4: 184–94. doi:[10.1111/2041-210X.12000](https://doi.org/10.1111/2041-210X.12000).

Luke, Sean. 2013. *Essentials of Metaheuristics*. Lulu, second edition, available at https://cs.gmu.edu/~sean/book/metaheuristics/Essentials.pdf. doi:[10.1007/s10710-011-9139-0](https://doi.org/10.1007/s10710-011-9139-0).