Altruist manual

Software for simulating biological evolution in structured and viscous populations

By Agner Fog

Last updated 2025-01-13

Introduction

This is a program for simulating biological evolution in structured or viscous populations. The program can simulate several different models of evolution, including various models of group selection, group territoriality, leadership, as well as a model where epistasis is leading to punctured equilibria. The simulation is based on neo-Darwinian theory with diploid organisms and discrete non-overlapping generations. The model can include one or more genetic loci with Mendelian inheritance.

The population is divided into groups. Each group can be confined to a fixed territory (an island) or a territory with floating boundaries. The rate of migration between islands or territories can be adjusted. Low migration rates can lead to interesting effects that can be studied by simulation.

The program keeps track of the gene pool of each group, but not necessarily the fate of each organism within a group. Random events such as growth, selection, mutation, migration, conflict, and group extinction are simulated by calculating the theoretical probability distribution for all possible outcomes and generating a random variate with the desired distribution to determine the actual outcome. All these random events contribute to genetic drift.

It is possible to watch evolution in progress. The program can show a map of all groups or territories with colors indicating the gene frequencies of each group.

All parameters can be modified by the user. The program can sweep parameter values automatically and repeat the simulation for each set of parameters in order to find the parameter ranges that lead to different outcomes, such as fixation of one allele or another, or stable polymorphism.

It is also possible to repeat a simulation multiple times with the same parameter values, but different random number seeds. This can be useful for investigating whether a result is reproducible, and for finding the average value of some result, for example the time it takes to reach fixation of a mutant gene.

Table of Contents

Getting started

Download the program package `altruist-win.zip` from github.com/AltruistSim/executable or the latest release from https://github.com/AltruistSim/source_code.

The program runs on a Windows computer with an x86-64 compatible microprocessor.

Extract the zip file to a separate folder. This folder will contain the executable file `altruist.exe`, and the library files `Qt6Core.dll`, `Qt6Gui.dll`, `Qt6Widgets.dll`, `platforms\qwindows.dll`, and a parameter file `last.altru`. You may add more parameter files.

Run the file `altruist.exe`. It may be necessary to tell your virus protection program to accept this file.

Open the menu File → Load parameter file, and select one of the example parameter files. Parameter files with different simulation examples can be found at github.com/AltruistSim/executable

Press Run → Start, or F5, to start the simulation. Watch the progress of the simulation on the screen.

The different simulation models are explained below.

User interface

Menu File

File → Load parameter file: This loads a set of parameters from a file `*.altru`.

File → Save parameter file: This saves the current set of parameters to a new file `*.altru`.

File → Set output file: This will save simulation results to a comma-separated file. Click on "Make data output file" and "Change file name". Any existing file with the same name will be overwritten. The file will be appended if you are doing multiple simulations without changing the file name and without closing the program. The file will be overwritten, rather than appended, if you click on "Change file name" again. If you close the program and open it again, then you have to specify a new file name if you do not want to overwrite the previous data file.

Mark which data you want in the output file. Some data are only available for certain models.

"Output interval" is the number of generations between each line of output. Set this to zero if you want only the final result of each simulation.

The output file is a standard comma-separated file that can be read by spreadsheet programs and text editors. Values in the file are separated by commas, while decimals are marked by a dot, regardless of the language settings of the computer.

Menu Model

Model: This menu point selects the basic simulation model and gives a short description. It does not set a suitable set of parameters for the model. This can be done instead by loading a parameter file.

The name of the last parameter file is shown for convenience. Note that the parameters may have been changed since this parameter file was last read or written.

Menu Parameters → Geography and migration

Geography and migration: This sets the number of groups, size of groups and new colonies, migration rate, and defines where migrants and colonizers come from. Each item may be present or absent depending on the model.

Maximum number of groups is the number of islands or the maximum number of group territories.

Carrying capacity is the maximum number of inhabitants in an island, or the maximum number or inhabitants per area unit.

Carrying capacity std. deviation. The carrying capacity of islands is normally distributed with this standard deviation, if not zero.

Colonization group size is the average population size of founder groups or new colonies.

Migration topology defines where immigrants and colonists can come from. Linear: each group has two neighbors. Two rows: each group has three neighbors. Quadratic: each group has four neighbors. Honeycomb: each group has six neighbors. Octal: each group has eight neighbors with diagonal migration possible. Random: migrants can go from any group to any other group regardless of distance. Common gene pool: Immigrants have the same average gene frequencies as the entire metapopulation.

Emigration pattern determines how the number of emigrants from a group may depend on group properties. Note that this does not determine the actual number of successful emigrants, only the potential for emigration. Possible values: "constant" means the potential for emigration is independent on the properties of the group. "proportional w. population" means that the potential for emigration is proportional to the population of the group. "prop. w. excess population" means the potential for emigration is proportional with the number of offspring in excess of the carrying capacity. "prop. w. group fitness * population" means the potential for emigration is proportional with the population of the group multiplied by the group fitness.

Immigration pattern determines where immigrants come from and what the number of migrants from one island to another depends on. For every receiving group, and for every generation, a source for the immigrants is chosen according to this pattern. Possible values: "from common gene pool" means that immigrants can come from anywhere. They are taken from a gene pool consisting of the entire metapopulation. "from random neighbor" means that a neighbor group is picked at random from the available neighbor groups defined by the migration topology. The mean number of immigrants is the migration rate multiplied by the emigration potential for the chosen neighbor group, as defined by the emigration pattern. "prop. w. population, from neighbor" means that immigrants come from a random neighbor group. The mean number of immigrants is the migration rate multiplied by the population of

the receiving group, and independent of the emigration potential or the neighbor group. This option can be used if the gene flow is mainly determined by exogamous mating. "prop. w. vacant capacity, from neighbor" means that immigration is only possible if the population of the group is below the carrying capacity. The mean number of immigrants is the migration rate multiplied by the unused carrying capacity of the receiving group. "from all neighbor groups" means that immigrants are mixed from all the neighbor groups defined by the migration topology. The mean contribution of each neighbor group is proportional to its emigration potential. The number of immigrants is the migration rate multiplied by the weighted mean emigration potential of all the neighbor groups. "from random inhabited island" means that migrants can go from any island to any other island regardless of geographic distance. All immigrants come from the same group, which is picked at random. The mean number of immigrants is the migration rate multiplied by the emigration potential of the chosen group. "from strong neighbor group" means that a neighbor group is picked in a biased way so that the probability of choosing a particular neighbor group is proportional to its emigration potential, as determined by the emigration pattern. The mean number of immigrants is the migration rate multiplied by the emigration potential for the chosen neighbor group.

Colonization pattern is similar to immigration pattern. The colonization pattern is used instead of the immigration pattern if a group has gone extinct or an island is empty. The size of the new founder population is determined by the “colonization group size” rather than the migration rate in this case. The actual size of the new colony is a random variable simulated by a Poisson distribution with the mean equal to the “colonization group size”. If this number exceeds the population of the migration source, then it is limited to the latter number.

Migration rate determines the mean number of immigrants to each island in one generation. The migration rate should normally be less than 0.5 because higher values may leave islands empty if everybody emigrates. The actual number of migrants is a random variable simulated by a binomial distribution with the mean calculated as explained above under “Immigration pattern”.

In models with floating territorial boundaries (group territoriality model and regality model), the mean number of immigrants to a territory is the migration rate multiplied by the carrying capacity of the territory multiplied by the size of the selected neighbor territory relative to the maximum size.

Menu Parameters → loci and mutation

Loci and mutation: This defines one or more genetic loci, Mendelian dominance, initial fraction of mutant genes, and mutation rates.

Menu Parameters → Individual and group fitness

Individual and group fitness: The parameters for individual fitness define the fitness of different combinations of phenotypes, selection model, and growth rate.

The group properties include parameters defining group fitness, war, extinction rate, and fraction of survivors after group extinction.

The group fitness function defines how the fitness of a group in relation to some group selection mechanism depends on the fraction of individuals with a certain mutant phenotype (altruism or regality). A convex curve may lead to stable polymorphism, while a concave curve more likely leads to fixation of either the wild type or the mutant gene. The curvature of the group fitness function can be expressed by the exponent, c:

$$G = f^c$$

where *G* is a measure of group fitness, and *f* is the fraction of individuals with the altruism or regality phenotype. A value of c below 1 gives a convex curve, a value of 1 gives a linear curve, and a value of c above 1 gives a concave curve. The setting "one is enough" (c = 0) means that a single mutant phenotype is enough to reach the full group advantage. "All or nothing" (c = ∞) means that there is no advantage unless all group members are mutant phenotypes.

These group properties are not present in all models. Only the parameters that are relevant for the selected model are present in the dialog box.

Menu Run

Run control: This sets the minimum and maximum number of generations to simulate, and a criterion for when to stop the simulation.

"Criterion degree" specifies a threshold for the stop criterion. If, for example, the stop criterion is that a certain gene locus is uniform and the criterion degree is 0.99 then the simulation will stop when at least 99% of the global gene pool is the same allele.

"Random generator seed" is an arbitrary integer number used for initializing the random number generator. Any integer value can be used. If you repeat a simulation with the same seed and the same set of parameters, you will get the same result. If you repeat a simulation with a different seed, you will get a different result because random events will have different outcomes. You may repeat a simulation with the same seed if you want to further study an interesting event. Or you may repeat the simulation with different seeds to see if events or results are reproducible.

"Delay" gives a delay after each generation so that the graphics on the screen is not updated too fast. If parameter sweeps are active, then you get a delay after each simulation instead.

"Parameter sweeps" will run multiple simulations with different values of a particular parameter. If multiple sweeps are defined, then the first one will be the innermost loop. "Linear" will make the parameter change from the start value to the end value with steps of equal length. "Logarithmic" will make the parameter go through a logarithmic scale. "Step" indicates the number of values per decade in this case.

"Linear search" and "Logarithmic search" will make the program vary the value of the parameter in order to search for the limits between values that lead to fixation of the wild type, fixation of the mutant, or stable polymorphism. The progress of a sweep or search is shown at a status line below the graphics. The result of a simulation is indicated with an "A" if the mutant allele is fixated, "E" if the wild type is fixated, or "P" if polymorphism remains after the specified maximum number of generations.

If both a search and a sweep is specified, then the screen will have a 2-dimensional map showing the simulation result of each set of parameters.

The following parameters are available for sweep. Floating point parameters are also available for search.

random seed. This will repeat the simulation with the same parameters, but a different seed, to see if results are reproducible, or for calculating mean and variance of result values.

migration rate. This is the relative rate of migration between groups, as determined by "immigration pattern".

group fitness curvature. This defines the curvature c of the group fitness as a function of the fraction of phenotypic altruists. See the explanation above.

altruist fitness. This defines the reproductive fitness of altruists. The value will be the same among egoists and among altruists.

fitness ratio. The fitness of altruists is set to the fitness of egoists multiplied by the fitness ratio. The value can be different among egoists and among altruists.

extinction rate for egoist groups. This sets the strength of group selection in an island model. The extinction rate for altruist groups will be unchanged.

war intensity. This sets the strength of territorial wars in a territoriality model or regality model.

max group size. Sets the maximum population in a group.

carrying capacity. Sets the carrying capacity of islands or the carrying capacity per area unit for a territoriality model or regality model.

colony size. Sets the size of new colonies or founder populations.

survival rate. Sets the relative number of survivors after group extinction or defeat in war.

haystack period. Sets the number of generations that a population stays isolated before they are mixed. Only in the haystack model.

fitness ratio among altruists/egoists. The fitness of each phenotype in groups dominated by altruists is calculated as the fitness of the same phenotype among egoists multiplied by this ratio.

Start: Start simulation.

Pause: Pause simulation.

Resume: Resume simulation after pause.

Single step: Run a single generation. Press this (or F8) repeatedly to run one generation at a time.

Stop: Stop simulation. It cannot resume after stop.

Screen output

The screen makes a graphic illustration depending on the model.

Island models and models with fixed territories are shown with a square for each island or group. The size of each square reflects the number of individuals in the group. The genetic composition of each group is indicated by a color. Models with a single locus have a blue color for the wild type and a red color for the mutant. Mixed groups are indicated by mixed colors, with a hue ranging from blue, over green and yellow, to red.

Models with multiple loci are shown with multiple colors: Black for the wild type, red for mutant allele at the first locus, blue for mutant allele at the second locus, and green for mutant allele at the third locus. All these colors can be mixed. White indicates all mutant alleles at all three loci.

You can click on any group to see the genetic composition and other properties of that group.

Models with group territories are shown with the shape and size of each territory with moving boundaries. The territories are colored in the same way as for island models.

The graphic image is updated after every generation during a single simulation. It is only updated at the end of each simulation when there are multiple simulations in a parameter sweep or search.

Another screen mode can show the results of a two-dimensional parameter search. The screen will show a two-dimensional map of parameter values. Areas that lead to fixation of the wild type are marked with blue, areas that lead to fixation of the mutant are red, while parameter ranges that fail to read fixation within the specified maximum number of generations are green. This screen mode will appear when the run control defines a parameter search of one parameter followed by a parameter sweep of another parameter. (The image may look distorted if the results are not consistently appearing in the same order: egoism-polymorphism-altruism, or the reverse).

Running time-consuming jobs

Some simulation jobs can take many hours, especially when using parameter sweep or search. You may have to prevent the computer from turning off automatically after some time. This is particularly important for laptop computers. In a Windows computer, go to power options, allow hybrid sleep, and set it to never hibernate.

If you have multiple time-consuming jobs, it is possible to run multiple instances of the Altruist.exe program simultaneously. You can efficiently run as many instances as the computer has CPU cores, but not more. You can distinguish the different instances by looking in the Model menu. Press Cancel after opening the Model menu to avoid disrupting an ongoing simulation. It shows the name of the last parameter file loaded or saved.

How the simulation works

A population of diploid organisms is divided into partially isolated groups (also called demes). The population goes through the stages of mutation, growth, selection, and migration between groups. Some

models include group extinction and recolonization of empty islands, or territorial war. These events are all stochastic and all contributing to genetic drift. The time dimension is defined by discrete, non-overlapping generations.

The program keeps track of the gene pool of each group, but not necessarily the fate of each individual organism. Random events such as growth, selection, mutation, drift, migration, intergroup conflict, and group extinction are simulated by calculating the theoretical probability distribution for all possible outcomes, and generating a random variate with the desired distribution to determine the actual outcome.

Statistical models

Almost everything in the simulated evolution is random. Simulation of random events is called Monte Carlo simulation. The results of random events are represented by stochastic variables with the relevant statistical probability distribution functions.

The program code includes a library of random variate generating functions with the desired probability distributions, such as uniform, normal, Poisson, binomial, hypergeometric, as well as various non-central hypergeometric distributions and multivariate distributions. These variates are generated using methods described by Stadlober[1] and Fog[2].

These random variate generating functions are based on a high quality pseudo-random number generator designed as a combination of a Mersenne twister and a multiply-with-carry generator[3]. A high quality random number generator is important for the reliable simulation of events or combinations of events that have very low probability.

Most events are simulated for each group separately.

Population growth may be simulated with a Poisson distribution giving the population size in each new generation. The Poisson parameter is the population size in the preceding generation multiplied by the growth rate.

The gene pool after growth may be simulated by a binomial distribution where the probability parameter for each allele is the fraction of this allele in the parent generation.

Mutation is simulated by a binomial distribution where the probability parameter is the mutation rate.

The number of organisms of each phenotype is determined from the gene pool by the Mendelian dominance at a biallelic locus. The two alleles at the locus are sampled by a hypergeometric distribution for each group.

---

[1] Stadlober, E. (1990). The ratio of uniforms approach for generating discrete random variates. *Journal of Computational and Applied Mathematics,* 31(1), 181–189.

[2] Fog, A. (2008). Sampling methods for Wallenius' and Fisher's noncentral hypergeometric distributions. *Communications in Statistics–Simulation and Computation,* 37(2), 241–257. https://doi.org/10.1080/03610910701790236

[3] Fog, A. (2015). Pseudo-Random Number Generators for Vector Processors and Multicore Processors. *Journal of Modern Applied Statistical Methods,* 14(1), 308–334.

Selection

Selection can be simulated by several different models:[4]

- **Fecundity selection:** Each phenotype has a different growth rate, proportional to its fitness. It is assumed that organisms are mating randomly, and possibly with multiple partners. The contribution of each phenotype to the gene pool of the next generation is a Poisson distribution. The number of offspring of one phenotype does not depend on what happens to other phenotypes.

  The population is subsequently limited to the carrying capacity of the habitat if the total number of offspring exceeds the carrying capacity. This limitation is simulated by sampling survivors by a hypergeometric distribution, where all phenotypes have the same probability of survival.

- **Positive viability selection:** This model is used when organisms compete for a limited resource, and different phenotypes have different chances of finding this resource. The group population size is limited by the resources available to the group. The fates of all organisms are interdependent in the sense that if individuals of one phenotype get more of the resource, then other phenotypes will get less. The probability distribution of survivors of each phenotype with interdependent fates is called Wallenius' noncentral hypergeometric distribution.[5]

- **Negative viability selection:** This model is used when organisms are selected sequentially for death rather than selected for survival. The population is limited by predation, and different phenotypes have different chances of avoiding predators. The number of predators is assumed to be constant due to a predator-prey equilibrium. The fates of all organisms are interdependent in the sense that if predators kill more of one phenotype, then they will kill less of other phenotypes. It is assumed that predators have limited appetite. The probability distribution of survivors in this model is called complementary Wallenius' noncentral hypergeometric distribution.[6]

- **Independent viability selection:** This is a model where the population grows first, followed by reduction in two steps. The first step involves selection where organisms of different phenotypes have different probabilities of surviving a particular danger, while their fates are independent. The population size is subsequently limited to the carrying capacity of the habitat in step two by a process where each phenotype has the same chance of survival. The probability distribution in the first step is a binomial distribution calculated with a probability parameter that is proportional to the fitness of each phenotype. The population reduction in step two is simulated by sampling the survivors from step one by a hypergeometric distribution. The independent viability selection model gives the same final probability distribution as fecundity selection if the

[4] Fog, A. (2023). Statistical models of different selection mechanisms. https://www.researchgate.net/publication/375029380_Statistical_models_of_different_selection_mechanisms

[5] Fog, A. (2008). Calculation methods for Wallenius' noncentral hypergeometric distribution. Communications in Statistics-Simulation and Computation, 37(2), 258–273. https://doi.org/10.1080/03610910701790269

[6] Same.

number of survivors from step one exceeds the carrying capacity of the habitat.

- **Minimum viability selection:** The positive and negative viability selection models, described above, both rely on interdependence of the fates of all organisms. The selection effect is smaller if the interdependence of fates is imperfect due to limited mobility. The minimum viability selection model represents a lower limit to the selection effect if interdependence is low due to low mobility. This effect is the same for positive and negative viability selection with low mobility. This model is simulated by Fisher's noncentral hypergeometric distribution.[7]

Migration

Migration can be simulated by several different models as described above under "migration topology", "emigration pattern", and "immigration pattern".

Simulation models

Island model

The Island model is the traditional model of group selection. The population of a species is divided into subpopulations, also called groups or demes. It is convenient to imagine that the subpopulations inhabit each their island, separated by geographical barriers, but other schemes can be found in nature. Social insects, for example, are organized as colonies each living in their own nest.

A group can go extinct. New groups are formed by members of a nearby group colonizing an empty island or forming a new nest in a vacant area.

The probability of extinction is a function of a property called "group fitness". The higher the group fitness, the lower the probability of extinction. The group fitness is a function of the number of phenotypic altruists in the group. Altruists are called so because their behavior is reducing their individual fitness, but increasing the group fitness. An example of altruistic behavior is to avoid overexploiting the resources available to the group. They are altruistic only towards their own group. This is called parochial altruism. Individuals without the altruism trait are called egoists.

The purpose of the simulation is to test whether group fitness can win over individual fitness so that the altruistic trait is spreading. We want to find the conditions under which this is possible.

The parameter menu for individual and group fitness specifies the following parameters:

Fitness of egoists and altruists. Egoists normally have higher fitness than altruists. It is possible to specify different fitness values depending on whether the individual is surrounded by egoists or altruists. Linear interpolation is used to calculate the fitness of an individual surrounded by a mixture of egoists and altruists.

[7] Fog, A. (2023). Statistical models of different selection mechanisms. https://www.researchgate.net/publication/375029380_Statistical_models_of_different_selection_mechanisms

The selection model can be fecundity selection or different kinds of viability selection. A growth rate is specified for viability selection only. The population is reduced to the carrying capacity of the island if growing above this limit.

The group fitness function specifies whether the group fitness as a function of the fraction of phenotypic altruists is a linear function or convex or concave. A convex function may lead to stable polymorphism, while a concave function is more likely to lead to fixation of either egoism or altruism alleles.

The extinction pattern determines whether the probability of extinction of a group depends only on the group itself or also on the fitness of neighbor groups.

The survival rate determines the mean fraction of individuals that are surviving after a group extinction. Any survivors will stay on the same island and mix with new colonizers from a neighbor island.

Extinction rate specifies the probability of extinction of a group in a single generation, depending on the group size and the group fitness. Intermediate values are calculated by linear interpolation.

An example set of parameters for this model is given in the file `island1.altru`.

The graphic display shows groups with egoism in blue and altruism in red. Intermediate groups have colors with a hue ranging from blue over green and yellow to red.

Conformity model

The conformity model is an extension of the island model. A gene named conformity at an extra locus with the same name controls a trait that enforces conformity.

Relative fitness for conformists specifies the fitness of phenotypic conformists versus non-conformists.

The fitness of altruists and egoists can be altered if there are many conformists in the group. A group with a high level of conformity can reduce the fitness of minority members or increase the fitness of majority members. For example, conformity may reduce the fitness of egoists in group dominated by altruists, or vice versa.

An example set of parameters for this model is given in the file `conform.altru`. Simulation shows that selection for conformity is weak. The conformity gene is mainly spreading *after* the altruism gene has become widespread.

The graphic display shows groups with conformity in green and altruism in red. Groups with both are yellow.

Outsider exclusion model

The outsider exclusion model is an extension of the island model. A gene named endogamy at an extra locus with the same name controls a trait that is reducing gene flow into the group.

Relative fitness for endogamists specifies the fitness of phenotypic endogamists versus exogamists.

The parameter menu for geography and migration has fields for specifying the rate of immigration with and without endogamy. Migration rates for intermediate levels of endogamy are calculated by linear interpolation.

An example set of parameters for this model is given in the file `endogamy.altru`. The endogamy allele can spread simply because it is reducing the competition from the exogamy allele. Simulation shows that the endogamy allele is spreading first and paves the way for subsequent spreading of the altruism allele.

This model is suitable for social insects and other eusocial species with communal nesting.

The graphic display shows groups with endogamy in blue and altruism in red. Groups with both are violet.

Haystack model

This model is a hypothetical scenario proposed by John Maynard Smith.[8] Imagine that groups of mice live in haystacks. Each haystack is colonized by a small founder population early in the season. Their descendants stay in the same haystack until the end of the season when the haystacks are removed. In a second period where there are no haystacks, the mice live elsewhere and the entire metapopulation is mixed perfectly until next season when new haystacks are colonized. There are two different phenotypes: altruists who limit their consumption of resources, and egoists who eat more than necessary. Haystacks with altruist mice leave more mice to the metapopulation at the end of the haystack season than haystacks with egoist mice. This mechanism is also called intrademic group selection.

The size of founder populations is specified as Colonization group size in the Geography and migration parameter box.

Other parameters are specified in the "Individual and group fitness" parameter box:
Mixing period is the number of generations in the mixing period where there are no haystacks.
Haystack period is the number of generations where groups are living isolated in haystacks.
Max metapopulation is the carrying capacity of the habitat in the mixing period.
Mix period growth rate is the growth rate in the mixing period.

The graphic display shows one big square in the mixing period and multiple smaller squares in the haystack period. Colors range from blue for egoism to red for altruism.

The haystack model is efficient if egoist groups are likely to go extinct, otherwise it is quite weak.

The parameter file `haystack.altru` contains an example of this model without group extinction.

[8] Smith, J. M. (1964). Group selection and kin selection. *Nature,* 201(4924), 1145–1147.

Group territoriality model

This is a new model introduced here. This model defines a species of social animals or humans living in groups. Each group has its own territory. A group can conquer territory from a neighbor group, possibly through violent conflict. The winning group will gain more territory, while the territory of the losing group gets smaller. The carrying capacity of a group territory is proportional to its area.

There are two phenotypes: altruists who are willing to fight for their group in territorial conflicts, and egoists who are not. Altruists have lower individual fitness than egoists because they may suffer injury or death in violent conflicts. But groups with many altruists have higher group fitness, which means higher ability to win territorial conflicts.

Groups with many altruists are likely to win territory from weaker neighbor groups. Larger territories can feed more group members and allow the groups to grow bigger. The population of a losing group will be reduced because the smaller territory can sustain fewer individuals.

When a group gets too small, it will be unable to defend its territory and lose it all to an attacking neighbor group. When a group gets too big, it will split into two groups that each gets half of the territory. The simulation code requires that group territories must be contiguous. If a territory happens to become noncontiguous, then the group will be unable to defend it and lose the smaller part.

This model involves two opposing evolutionary forces. Group selection will favor the gene for altruism because groups with many altruists can grow larger by conquering new territory. But individual selection favors the opposite allele, egoism, that increases the survival rate or reproduction rate of the individual.

The purpose of the simulation is to test whether group fitness can win over individual fitness so that the altruistic trait is spreading, and to find the conditions under which this is possible.

The strength of a group is defined as the group fitness multiplied by the number of individuals in the group. The higher the difference in strength between two groups, the more territory can the stronger group win from the weaker group.

The parameters that can be specified for this model include the following:

Total area (under Geography and migration) is the area of the entire habitat in arbitrary area units. The program may change the value to the nearest square number.

Carrying capacity is the maximum number of individuals per area unit. The carrying capacity may be the same for groups of egoists and groups of altruists, or it may depend on the composition of the group.

Max. territory area is the maximum area of a group territory. Groups with more territory than this will split up.

Min. territory area is the minimum area that a group can survive in and defend.

Migration rate is the rate of migration between neighbor groups.

War pattern (under Individual and group fitness) defines different models for territorial conflicts:
War against all makes each group attempt to attack all its neighbor groups.
Depends on shared border length makes each group attack one neighbor group per generation. The probability that it selects a particular neighbor group for attack is proportional to the length of the

border shared with that group.
Depends on difference in strength makes each group attack one neighbor group per generation. The probability that it selects a particular neighbor group for attack is inversely proportional to the strength of that group.

Survival degree specifies how many members of a losing group that are incorporated into a winning group. The probability that a member of a losing group is entering the winning group is the survival degree multiplied by the fraction of territory that is lost to the winning group.

War intensity. The expected amount of territory that a group can conquer from a neighbor group is proportional to the war intensity multiplied by the difference in group strength.

The graphic display shows all territories and boundaries. Colors range from blue for egoism to red for altruism. You can click on a group territory to see the area, population, and gene pool for the group.

The parameter file `terri.altru` gives an example of this model.

Regality model

The regality model is similar to the group territoriality model, but based on leadership rather than altruism. Fighting for one's group involves a collective action problem when the cost of fighting is borne by the individual fighter while the benefit resulting from the actions of this individual is divided between all group members. Group selection can explain such altruistic behavior only when the rate of migration between groups is extremely low. Otherwise, the group needs an effective organization and coordination in order to overcome the collective action problem.

A theory called regality theory[9] explains how this can be achieved through strong leadership. Regality theory is developed to explain human behavior in war. It has not yet been investigated whether similar mechanisms can explain intergroup conflict in other social animals.

Humans have evolved a psychological response pattern that makes us support a strong leader in case of war or perceived collective danger, according to regality theory. The leader can compensate individuals for the cost of fighting by rewarding brave warriors and punishing cowards and defectors.

There is a high fitness advantage in being a powerful leader. For example, a male leader may use his power to get multiple female partners.[10] This fitness advantage compensates for the cost of leading. The leader has a strong incentive to strengthen the group because of the fitness advantage of being the leader of a large group.

The simulation model will test whether collective fighting can be explained by individuals supporting a strong leader. This model works as follows:

---

[9] See www.regality.info/.

[10] A powerful male leader can gain fitness by having multiple female partners, while female leaders cannot gain much fitness by having multiple male partners. Therefore, males are willing to invest more than females in their attempt to become leaders and to exercise their leadership well. This is an important reason why, historically, most war leaders have been men. See Garfield, Hubbard, & Hagen: Evolutionary Models of Leadership. *Human Nature,* 30(1), 23–58, 2019.

A gene that we will call regality at a locus with the same name makes individuals support a leader who can organize collective fighting in intergroup conflict. The power of the leader is proportional to the fraction of group members who support him. This gives the leader more power and fitness at the cost of all other group members who will have less fitness. It is important to note that the loss of fitness applies equally to all non-leaders, regardless of their genes. The regality allele does not involve an increased willingness to fight, but instead a willingness to a support a leader who can make everybody fight, including those who do not have this allele.

The strength of a group is calculated as the size of the group multiplied by the power of the leader. In other words, a large group with many regality members will have more strength to conquer territory from weaker neighbor groups.

The fitness consequences of conquering new territory from a weaker neighbor group or losing territory to a stronger neighbor group are obvious. The consequences of selection inside the group, however, are complicated and difficult to understand.

To explain this, we will first analyze the situation with the territory size kept constant. We are assuming that the selection of a leader is unbiased so that individuals with different genotypes have the same probability of becoming a leader. This means that there is no direct selection acting on individuals. The only selection is an indirect selection by the consequences for each individual of its own contribution to group-level phenomena.

Assume that the relative fitness of the leader is $1 + f\Lambda$, where *f* is the frequency of the regality phenotype in the group, and Λ is a factor called leader advantage. We are assuming a haploid species here in order to simplify the argument, even though the simulation code involves a diploid species.

When an individual with the regality phenotype supports the leader, the behavior of this individual makes an incremental contribution to the fitness of the leader of the size Λ/N, where N is the number of individuals in the group. The fitness of all non-leaders is decreased by the same amount, shared between the *N*-1 non-leaders. Thus, the individual with the regality allele suffers a fitness loss of $\frac{\Lambda}{N(N-1)}$ as a consequence of its own support for the leader. The mean number of non-leaders with regality is $(N-1)f$ which makes the expected total loss of regality alleles equal to $f\Lambda/N$.

The consequence of the behavior of one individual non-leader with regality phenotype to other non-leaders is irrelevant because this individual does not know the genotypes of anybody else. Transferring fitness from other individuals with unknown genotypes to a leader with unknown genotype has no net effect as long as the leader has the same probability (*f*) of having the regality allele as the other group members.

Here, we have assumed that the fitness gain of the leader is proportional to the number of group members with the regality phenotype, including the leader himself. In other words, the leader is increasing his own fitness by Λ/N if he has the regality allele. The probability that the leader has regality is *f*, so that the mean increase in regality due to the leader supporting himself is $f\Lambda/N$. This results in a positive selection for the regality allele by an amount that is exactly equal to the total negative selection for all the non-leaders. The net result is no selection for or against the regality allele at the group level in the absence of war. This result applies also for diploid inheritance as long as the competition for fitness is a zero-sum game and the chance of becoming a leader is independent of the regality allele.

This result is counterintuitive. We would expect negative selection against a gene that makes individual group members transfer some of their fitness to a despotic leader. This apparent flaw in the model comes from the assumption that the leader supports himself only if he has the regality phenotype. We can remove this effect by assuming that the leader will support himself out of self-interest regardless of whether he has the regality phenotype or not. This will remove the dependence of the leader's power on his own genes and thereby remove the positive selection for the regality allele, while the negative selection remains.

The simulation model defines a parameter named leader regality counts in the Individual and group fitness dialog box. Setting leader regality counts to true means that the leader behaves differently depending on his phenotype, with the consequence that positive and negative selection for the regality allele cancels out in the absence of war. Setting leader regality counts to false means that the leader behaves the same way, regardless of his genotype or phenotype, with the consequence that there is a (weak) selection against the regality allele in the absence of war. Leader regality counts must be false if the regality allele has incomplete dominance.

A parameter named leader advantage is the Λ value in the above explanation. A positive value gives the leader extra fitness. A negative value gives the leader less fitness than others.

Leader selection is 1 by default. A value higher than 1 means that individuals with the regality phenotype have an increased chance of becoming a leader. A value lower than 1 means that they have a decreased chance of becoming a leader.

The values of fitness of neutrals and regalists indicates the fitness of individuals without and with the regality phenotype. These values should be equal in the default model. A situation where the leader can reward those who support him and punish those who don't can be simulated by setting the fitness of regalists among regalists higher than the fitness of neutrals among regalists.

All other simulation parameters are the same as for the group territoriality model, as described above.

The graphic display shows all territories and boundaries. Colors range from blue for neutral to red for regality. You can click on a group territory to see the area, population, and gene pool for the group.

The parameter file `regality.altru` gives an example of this model.

Epistasis model

The purpose of this model has nothing to do with the previous models. This model illustrates an example of how epistasis[11] can lead to evolution through punctuated equilibria.

Imagine a species where a new mutant gene, A, is decreasing fitness. Another new mutant gene, B, at a different locus is also decreasing fitness. But fitness is increased when both mutant genes A and B are present in the same individual organism. This model explores the possibility that the AB combination will appear by chance and then spread in the population.

[11] Epistasis means interaction between genes at different loci. The effect of a gene at one locus depends on a gene at some other locus.

If the AB combination is starting to spread through the population, then another mutant gene, C, at a third locus can increase the fitness further and result in a new variant ABC.

This model illustrates the phenomenon of evolution through punctuated equilibria. The rare event of A and B being combined in the same individual organism can start a new course of evolution that moves the population from one peak in the fitness landscape to a new higher peak. Other genes at other loci will then adapt to the new situation and make the species climb further up the new peak. The latter event, illustrated by the transition from AB to ABC, can be much faster than the transition from the wild type to AB if it requires only a single mutation to increase the fitness by going from AB to ABC. The consequence is that the intermediate form AB will occur in small numbers during the course of evolution, while the final form ABC will be much more numerous. This can explain why "missing links" or intermediate forms are rarely found in the fossil record.

The initial combination of A and B is more likely to occur in a viscous population or in small relatively isolated patches where genetic drift is high. This viscosity or isolation is simulated by dividing the population into small islands or patches with a limited rate of migration between these.

The parameter menu Individual and group fitness is used for specifying the fitness of each combination of two different phenotypes at each locus. There are four possible combinations if locus A and B are used, and eight possible combinations if all three loci, A, B, and C, are used.

The graphic display shows each group or subpopulation with colors indicating the alleles. The neutral wild type is black, mutants A are red, mutants B are blue, and mutants C are green. The colors can be combined so that the AB variant is violet and ABC is white. Mixed groups have mixed colors.

You can click on any group to see its gene pool.

The parameter file `epistasis.altru` gives an example of this model.

Source code documentation

Simulating random events

Computer simulation of a sequence of random events is called Monte Carlo simulation. The randomness is driven by a pseudo random number generator. The word "pseudo" indicates that the random sequence is deterministic and reproducible. The random number generator is initialized by a so-called seed. The seed can be any integer number. If a simulation is repeated with the same seed and exactly the same parameters, then the result will be the same. A different seed will give a different sequence. This is exactly what we need in the simulation of evolution. You can repeat a simulation with the same seed if you want to reproduce it and study an interesting event further. And you can repeat it with different seeds to see if the observed event is a common occurrence or a rare aberration.

Older simulation programs have often used random number generators with poor quality. The present program is using a random number generator of very high quality. This is necessary in order to reliably

simulate rare events or combinations of events that have very low probability. The random number generator used here is a combination of a Mersenne twister and a multiply-with-carry generator.[12]

The random number generator is generating random numbers with uniform distribution. This can be converted into random variates with other statistical distributions.

Standard distributions such as the normal distribution, binomial distribution, hypergeometric distribution, Poisson distribution, etc. are generated using methods described by Stadlober.[13] Some special distributions relating to natural selection include Wallenius' non-central hypergeometric distribution,[14] which is generated with methods developed by Fog.[15]

The basic random number generator and functions for generating the standard distributions are defined in the file `random.cpp`. Wallenius' non-central hypergeometric distribution and other special distributions are defined in the file `wallenius.cpp`.

Generation loop

The simulation is going through loops for each generation and each subpopulation (group or island). It is assumed that generations are discrete and nonoverlapping. The code keeps track of the gene pool and certain other properties of each group, but not individual group members. The following events may be simulated inside the generation loop:

Mutation. The probability that a gene will mutate is the mutation rate μ. The number of mutations of g_a genes is simulated with the distribution $mutations \sim \text{binomial}(g_a, \mu)$.

Migration. Migration of individuals from a neighbor group into a particular group is simulated by first choosing a random neighbor group according to the specified immigration pattern. The probability that an individual will migrate is the migration rate r. The number of migrants going from one group to another is simulated with the distribution $migrants \sim \text{poisson}(n\, r)$, where n is the number of potential migrants.

Reproduction and growth. The number of offspring of a group of n individuals is simulated by the Poisson distribution. $offspring \sim \text{poisson}(n\, g)$, where g is the growth rate.

Gene pools versus genotypes. Consider a locus with two alternative genes denoted “a” and “A”. The gene pool of a group with n individuals is recorded as the number of each allele, g_a and g_A. The three possible genotypes, (aa), (aA), and (AA) may have different fitness. Therefore, we need to split the gene pool into genotypes before we can simulate selection. This done by drawing allele randomly from the gene pool for each chromosome:

---

[12] Fog, A. (2015). Pseudo-Random Number Generators for Vector Processors and Multicore Processors. *Journal of Modern Applied Statistical Methods,* 14(1), 308–334.

[13] Stadlober, E. (1990). The ratio of uniforms approach for generating discrete random variates. *Journal of Computational and Applied Mathematics,* 31(1), 181–189.

[14] Fog, A. (2008). Calculation methods for Wallenius’ noncentral hypergeometric distribution. *Communications in Statistics - Simulation and Computation,* 37(2), 258–273.

[15] Fog, A. (2008). Sampling methods for Wallenius’ and Fisher’s noncentral hypergeometric distributions. *Communications in Statistics - Simulation and Computation,* 37(2), 241–257.

$n_{a\_} \sim \text{hypergeometric}(n, g_a, 2n)$

$n_{aa} \sim \text{hypergeometric}(n_{a\_}, g_a - n_{a\_}, n)$

$n_{aA} = g_a - 2n_{aa}$

$n_{AA} = n - n_{aa} - n_{aA}$,

where $n_{a\_}$ is the number of individuals with allele "a" at the first chromosome, n_{aa} is the number of individuals with allele "a" at both chromosomes, n_{aA} is the number of heterozygotes, and n_{AA} is the number of individuals with allele "A" at both chromosomes. (This process is coded in function `combineGenes` in the file `run.cpp`).

The three different genotypes represent two different phenotypes if allele "A" is recessive or dominant, or three different phenotypes in case of incomplete dominance.

Selection. There are several different possible selection models:

Fecundity selection means that each phenotype has a different growth rate equal to the specified reproductive fitness. The number of offspring from each genotype is simulated as a Poisson distribution:
$offspring\ from\ genotype\ x \sim \text{poisson}(number\ of\ x\ \cdot growth\ rate\ for\ x)$
The Poisson process is simulated separately for each genotype. Recognizing that the sum of multiple Poisson variates is also a Poisson variate, we can see that the total number of offspring follows a Poisson distribution with a mean equal to the number of individuals in the parent generation multiplied by the mean growth rate.

Positive viability selection is the situation where the population is limited by resources and different phenotypes have different chances of finding this resource. The vector of genotypes in the offspring generation is simulated by drawing from a multivariate Wallenius' noncentral hypergeometric distribution:

$$\boldsymbol{g}_{offspring} \sim multiWalleniusNoncentralHypergeometric(\boldsymbol{g}_{parent}, \boldsymbol{fitness}, n)\ ,$$

where $\boldsymbol{g}_{parent}$ is the vector of genotypes in the parent generation, $\boldsymbol{fitness}$ is the vector of relative fitness of each genotype. Each vector has three elements for the three possible genotypes.

Negative viability selection is the situation where the population is limited by predation, and different phenotypes have different chances of avoiding predators. This is simulated in the same way as positive viability selection where a complementary multivariate Wallenius' noncentral hypergeometric distribution is used instead.

Minimum viability selection specifies a lower limit to the selection effect of positive or negative viability selection where the interdependence of the fates of all individuals is imperfect due to limited mobility. This is simulated with a multivariate Fisher's noncentral hypergeometric distribution.

Independent viability selection is a situation where different phenotypes have different chances of surviving a particular danger, while the fates of individuals are independent, and the number of survivors may exceed the carrying capacity of the group's territory. The number of survivors of genotype x is simulated as

$$survivors_x \sim binomial(n_x, fitness_x)$$

If the number of survivors exceeds the carrying then the population is reduced to the carrying capacity as described below.

Population limitation. If the number of individuals remaining after growth, selection, and migration exceeds the carrying capacity of the group's territory, then the population is reduced to the carrying capacity, K, in a way where all phenotypes have equal chance of survival, using a multivariate (central) hypergeometric distribution:

$$\boldsymbol{g}_{survivors} \sim multiHypergeometric(\boldsymbol{g}_{offspring}, K)$$

Each vector has three elements for the three possible genotypes.

Genetic drift is mainly the result of randomness in all of the above processes. It is assumed that individuals are mating randomly, and possibly with multiple partners. The mean contribution of each genotype to the gene pool of the next generation is proportional to its fitness. Heterozygotes do not necessarily contribute an equal amount of each allele to the offspring generation, but an amount simulated by a binomial distribution with probability parameter 0.5.

The simulation algorithm is following gene pools or genotype numbers rather than individuals. It does not decide who is male and who is female, and it does not take sex distribution and nonrandom mating patterns into account.

Group fitness. This variable is computed in models with any form of group selection or group conflict. The group fitness G is a linear or nonlinear function of the fraction of individuals with altruism or regality phenotype, as described above under the Individual and group fitness menu, $G = x^c$.

Group extinction. Groups can go extinct with a certain probability in models where group extinction is used. The probability that a group goes extinct in a certain generation is a linear function of the group fitness and the size of the group relative to the carrying capacity of the island or group territory.

The extinction rate is specified for big and small groups with minimum group fitness (egoists) and maximum group fitness (altruists). The extinction rate for groups with intermediate group fitness is calculated by linear interpolation between these values.

A few members of an extinct group may survive if the parameter survival degree is not zero. The number of survivors is simulated as a binomial distribution, where the probability of survival is the specified

survival degree. Survivors are picked at random from the gene pool of the group. The survivors are mixed with the new colonizers. For models with floating territories, the survival degree multiplied by the fraction of area lost determines the probability that a member of the losing group is incorporated into the winning group.

Recolonization. Empty islands are colonized immediately after extinction. The colonizers come from a neighbor group, a random group, or a common gene pool, as specified by the parameter Colonization pattern. The number of colonizers is simulated as a Poisson distribution. The mean is the parameter Colonization group size. The number of colonizers cannot exceed the number of inhabitants in the island that the colonizers come from.

The genes of the colonizers are picked at random from the gene pool of the group where the colonizers come from, using a hypergeometric distribution. These gene counts are added to the new island and subtracted from the island where the colonizers come from.

Territorial war. The habitat is laid out as a quadratic grid with arbitrary area units. Each group is given a territory of approximately equal size at the beginning of the simulation. A territory can have any number of neighbor territories, depending on shapes.

Every generation, each group tries to attack one or more neighbor groups, according to the defined war pattern. The strength of each group is computed as the group fitness multiplied by the number of group members. The expected gain, in area units, is computed as the difference in strength between the attacking group and the attacked group, multiplied by the specified war intensity, and divided by the carrying capacity per area unit.

The actual gain is simulated as a normal distribution with the expected gain as mean. The standard deviation is currently defined as the war intensity times the maximum territory size times 0.1.

If the actual gain is positive, then an area with this size is transferred from the neighbor group to the attacking group. Negative gains are ignored for technical reasons. Instead, a transfer of area in the opposite direction may happen if the neighbor group attacks the current group.

The transferred area points are chosen with the shortest possible distance from the center of the winning group and farthest from the center of the losing group. Group territories must be contiguous. A non-contiguous territory cannot be defended. If a territory happens to become non-contiguous, then the smallest part is lost to the attacker. Enclaves can exist and can be defended.

The carrying capacity of a territory is proportional to its area. A group that has won a piece of territory will grow in numbers until it reaches the carrying capacity on the expanded territory. A group that has lost a piece of its territory will be reduced in numbers to the carrying capacity of the reduced territory.

Some members of a losing group may survive and be adopted into the winning group if a survival rate bigger than zero is specified. The probability that an individual will go into the winning group is calculated as the survival rate times the fraction of area lost. The actual number of survivors is simulated as a binomial distribution with this probability parameter.

If a territory has grown bigger than the specified maximum area, then it is split into two territories of equal size and the population is divided randomly between the two new territories.

Program structure

The Altruist program is coded in C++ for the sake of performance and portability. Performance is important here because of the heavy calculations.

The graphical user interface is using the Qt framework for the sake of portability. The program is currently complied for Windows only, but it should be possible to recompile it for other platforms. The current version is compiled with Microsoft Visual Studio 2022, but other compilers and IDEs can be used as well. The source code is available at github.com/AltruistSim under a GNU General Public License.

To run under other platforms than Windows, it is necessary to compile the C++ source code. Include the Qt graphic framework version 6.4 or higher, and compile for the C++17 standard or higher.

User interface thread and worker thread

The program is running two threads. The main thread takes care of the user interface, including dialog boxes for setting parameter values, input and output of parameter files, and graphic representation of ongoing simulations and results.

The heavy calculations are done in the worker thread in order to keep the user interface responsive while a simulation is going on. The worker thread is encapsulated by the class `Worker` and initialized by the main thread.

The file `altruist.cpp` contains the program entry and initialization of the worker thread. The main thread can start a simulation by sending a signal to the worker thread, using the signal/slot mechanism in the Qt framework. The worker thread sends a signal back to the main thread when a simulation is finished.

The worker thread is paused while a graphic image is being updated in order to avoid changing data before the drawing is finished. The volatile Boolean variable `requestUpdate` takes care of this synchronization.

The worker thread also takes care of outputting simulation results to a comma-separated data file.

The variable `runState` indicates the state of a running simulation. Possible values include `state_idle`, `state_start`, `state_run`, etc.

The variable `sweepState` indicates the state of running parameter sweeps and searches.

Error handler

Errors can happen in all threads, but error messages in the form of popup messages can only be created in the main thread because this is the only thread that has access to the graphical interface. Typical errors are parameter values out of range in some function.

Errors are handled by a global object named `errors` of class `ErrorReporter`. This object is accessible to all parts of the program.

An error is reported by calling `errors.reportError(text)` from anywhere in the program. The `errors` object is checked inside the message loop of the main thread to see if any error message has been generated. The main thread will then create a message box with the error message if an error has been detected.

Data structures

The big data structure `AltruData` includes all shared data of a simulation, including parameters, running data, and results. `AltruData` is declared in the file `altruist.h`. There is only one instance of `AltruData`. It is defined in the class `Altruist`, which represents the main program window. The `AltruData` object named `d` is shared through pointers with all parts of the program.

Each model has its own Group structure with a unique name. The Group structure contains the necessary data to describe a single group, such as the number of inhabitants, carrying capacity, gene pool, and various group properties. For example, the Island model defines a structure named `IslandGroup`. A dynamic array `groupData` is allocated with a size big enough to contain the Group data for the maximum number of groups or islands. The function `Altruist::run()` in file `run.cpp` takes care of the allocation of `groupData`.

Any additional memory buffers needed by each model are allocated by the code for that model. The pointer array `extraBuffer[]` contains pointers to all such additional memory buffers.

All memory buffers are recycled when parameters or models are changed, as long as they are big enough to contain the new data. They are re-allocated only if the size is insufficient.

All allocated memory is deleted by the destructor for the class `Altruist` In the file `altruist.cpp`.

As the Group structure may be different for each model, the main program needs information about the size of this structure and the type and position of its fields for the current model. This information is needed for graphic display during a simulation and possibly for result statistics. Information about the De Group me structure of the current model is contained in an array of `GroupFieldDescriptor` structures. There is one such array for each model. The `GroupFieldDescriptor` array contains information about the size of the current Group structure, as well as the type and position of each structure member.

We are not using C++ member pointers for indicating the position of each structure member because the binary representation of member pointers may depend on the compiler, and because it is problematic to cast a member pointer of one class to a member pointer of another class. Instead, we are using the address offset of each data member relative to the beginning of the structure. This method is sure to be portable.

Other important data structures include the following:

`SweepParameter`. Name and position of a parameter in an automatic parameter loop or search. Declared in `parameterloop.h`.

`ParameterLimits`. Record in list of search results used for 2-dimensional map of parameter limits. Declared in `parameterloop.h`.

`ResultSet`. Record in a list of simulation results used for parameter search. Declared in `parameterloop.h`.

`TPoint`. (x,y) point used in manipulation of group territories.

`TPointDist` inherits `TPoint`. Used for sorting points by distance.

`IslandGroup`. A group structure used in island model.

`HaystackGroup`. A group structure used in haystack model.

`TerriGroup`. A group structure for group territories with floating boundaries. Used in group territoriality model and regality model.

`EpistasisGroup`. A group structure used in epistasis model.

Model code files

The program can simulate several different models. Each model is defined in a separate C++ file, such as `island_model.cpp`, `territoriality_model.cpp`, etc.

Model files can be added or removed freely to the C++ project without modifying the rest of the program code. Each model file contains a global object of the class `Construct`. The constructor for class `Construct` takes care of registering all necessary information about the model to a common list of model descriptors. This mechanism makes it possible to add or remove models simply by adding or removing model files.

The list of model descriptors is contained in the global object `models` of class `ModelDescriptorList`. This is an array of structures `ModelDescriptor`. The model descriptor for each model contains the name of the model, a descriptive text, a pointer to the array of `GroupFieldDescriptor` objects, a pointer to an array of structure `ParameterDef` defining non-standard parameters to appear in the dialog boxes, a pointer to an initialization function, and a pointer to a generation function.

Dialog boxes

The user interface can display a number of dialog boxes where the user can enter values of relevant parameters. These dialog boxes are generated dynamically to show only fields that are relevant to the selected model.

The initialization function for each model defines which parameter fields to show in the dialog boxes by setting a bit for each parameter that is relevant to the model.

The integer `bGeographyParametersUsed` in the `AltruData` structure has one bit for each field in the “Geography and migration” dialog box.

`bImmigrationPat` defines which immigration patterns can be selected in the “Geography and migration” dialog box.

`bEmigrationPattern` defines which emigration patterns can be selected in the “Geography and migration” dialog box.

`bTopology` defines which migration topologies can be selected in the “Geography and migration” dialog box.

`bRecolPat` defines which colonization patterns can be selected in the “Geography and migration” dialog box.

`bGroupPropertiesUsed` has one bit for enabling each group property in the “Individual and group fitness” dialog box.

`bExtinctionPatterns` defines which group extinction patterns can be selected in the “Individual and group fitness” dialog box.

`bWarPatterns` defines which war patterns can be selected in the “Individual and group fitness” dialog box.

`nLoci` specifies the maximum number of different gene loci that the model allows.

`locusUsed` specifies which of the `nLoci` loci are enabled by the user.

`bStopCriterionUsed` specifies which stop criteria can be specified in the “Run control” dialog box.

`graphicsTypeForModel` specifies what kind of graphic display fits the model.

It is possible to make additional dialog fields for model-specific parameters by defining an array of `ParameterDef` structures in the model file. Each record defines the type, name, and array size of a model-specific parameter or array of parameters. The model-specific parameters are normally stored in the `modelspec_i` or `modelspec_f` array in `AltruData`. The model specific parameters will be accessible in the “Individual and group fitness” dialog box.

Model-specific variables that are used during simulation or for results, but not for input parameters, can be stored in the `userData` array in `AltruData`. `bUserDataType` indicates for each element in `userData` whether it is integer or float. Names can be stored in an array pointed to by `userDataNames`.

List of source files

altruist.h Basic header file with declaration of the basic structures, classes, and constants of the whole program.

menus.h Declares classes, functions, and objects for user interface menus and dialog boxes with dynamic contents.

parameterloop.h Declares structures, classes, and constants for automatic parameter loops and parameter searches.

graphics.h Declares the class `AltruistView` and various constants for graphic representation of simulation results.

habitat.h Declares classes, structures, and functions for manipulating territories of arbitrary size, mapping territory points, walk around the border of a territory, and transferring area points from one territory to another. This is used in the group territoriality model and regality model.

random.h Declares classes and functions for pseudo random number generators and random variate generation.

ui_altruist.h Autogenerated by the Qt user interface framework.

stdafx.h Includes all the header files. Used by compilers that generate precompiled headers.

altruist.cpp Main program entry and initialization of the main window and important data.

menus.cpp Makes user interface menus and dialog boxes with dynamic contents.

run.cpp Basic functionality for the worker thread: starting, stopping, pause, single-stepping, check stop criterion, and some gene manipulation functions.

parameterloop.cpp Automatic parameter loops and parameter searches.

parameter_file_io.cpp Reading and writing `*.altru` files defining a simulation model with all its parameter values.

data_file_out.cpp Write comma-separated files `*.csv` or `*.txt` with simulation results.

graphics.cpp Graphic representation of simulation results in the form of graphs and geographic maps.

habitat.cpp Functions for manipulating territories of arbitrary shape, mapping territory points, walk around the border of a territory, and transferring area points from one territory to another. Used in group territoriality model and regality model.

random.cpp Pseudo random number generators and functions for generating random variates with standard distributions, such as uniform, normal, Poisson, binomial, hypergeometric, etc.

wallenius.cpp Functions for generating random variates with Wallenius' and Fisher's noncentral hypergeometric distributions.

stdafx.cpp Used by compilers that generate precompiled headers.

island_model.cpp Model file for island model, conformity, and endogamy models.

haystack_model.cpp Model file for haystack model, also called intrademic group selection.

territoriality_model.cpp Model file for group territoriality model with floating territorial boundaries.

regality_model.cpp Model file for regality model with leadership and floating territorial boundaries.

epistasis_model.cpp Model file for epistasis model simulating evolution through punctuated equilibria.

altruist.qrc Qt resource file

altruist.ui Qt user interface file

Altruist.vcxproj Visual Studio 2022 project file

Altruist.sln Visual Studio 2022 solution file

Altruist-win.zip Executable file and necessary library DLLs.

List of classes

name	base class	header file	code file
`Altruist`	`QMainWindow`	`altruist.h`	`altruist.cpp`

Main window, control of user interface. Sends signal to worker thread to start and stop simulation.

name	base class	header file	code file
`AltruistClass`		`ui_altruist.h`	`ui_altruist.h`

User interface setup. Autogenerated by Qt.

name	base class	header file	code file
`ModelDescriptorList`		`altruist.h`	`altruist.cpp`

List of installed simulation models.

name	base class	header file	code file
`Construct`		`altruist.h`	`altruist.cpp`

The constructor of this class adds a model to the model descriptor list.

name	base class	header file	code file
`ErrorReporter`		`altruist.h`	`altruist.cpp`

Handles error messages from all parts of the program. Causes a popup window with an error message.

name	base class	header file	code file
Worker	QObject	altruist.h	run.cpp

Does the simulation work in the worker thread. Receives commands from the main thread through doWorkSlot. Sends messages to the main thread through resultReadySignal. Calls functions in the selected model. Includes random number generator.

name	base class	header file	code file
ParameterLoop		parameterloop.h	parameterloop.cpp

Define an automatic parameter loop or parameter search. Has one instance for each loop.

name	base class	header file	code file
Habitat		habitat.h	habitat.cpp

Keeps track of points on the geographic map of floating territories. Finds owners and neighbors.

name	base class	header file	code file
SortedPointList		habitat.h	habitat.cpp

Sorted list of points used in function Habitat::splitArea.

name	base class	header file	code file
SortedPDList		habitat.h	habitat.cpp

List of points sorted by distance. Used in function Habitat::conquer.

name	base class	header file	code file
RandomVariates	RandomCombined	random.h	random.cpp

Random number generator, including random variates with different distributions.

name	base class	header file	code file
RandomCombined	RandomMersenne, RandomMother	random.h	random.cpp

Combines two different random number generators in order to improve randomness.

name	base class	header file	code file
RandomMersenne		random.h	random.cpp

Random number generator of the type Mersenne twister.

name	base class	header file	code file

RandomMother		random.h	random.cpp

Random number generator of the type multiply-with-carry.

name	base class	header file	code file
WalleniusNCHypergeometric		random.h	wallenius.cpp

Methods for calculating the univariate Wallenius' noncentral hypergeometric probability function.

name	base class	header file	code file
MultiWalleniusNCHypergeometric		random.h	wallenius.cpp

Methods for calculating the multivariate Wallenius' noncentral hypergeometric probability function.

name	base class	header file	code file
MultiWallenius-NCHypergeometricMoments	MultiWallenius-NCHypergeometric	random.h	wallenius.cpp

Calculates the exact mean and variance of the multivariate Wallenius' noncentral hypergeometric distribution.

name	base class	header file	code file
FishersNCHypergeometric		random.h	wallenius.cpp

Methods for calculating the univariate Fisher's noncentral hypergeometric probability function.

name	base class	header file	code file
MultiFishersNCHypergeometric		random.h	wallenius.cpp

Methods for calculating the multivariate Fisher's noncentral hypergeometric probability function.

name	base class	header file	code file
QMenu	QWidget	<QMenu>	

Qt pull down menu.

name	base class	header file	code file
QGridLayout	QLayout	<QGridLayout>	

Place dialog box widgets in a grid.

name	base class	header file	code file
QLabel	QFrame	<QLabel>	

Text label in dialog box.

name	base class	header file	code file
QLineEdit	QWidget	<QLineEdit>	

Edit field for entering values in dialog box.

name	base class	header file	code file
QPushButton	QAbstractButton	<QPushButton>	

Pushbutton in dialog box.

name	base class	header file	code file
QComboBox	QWidget	<QComboBox>	

Combo box in dialog box.

name	base class	header file	code file
QCheckBox	QAbstractButton	<QCheckBox>	

Checkbox in dialog box.

name	base class	header file	code file
ModelDialogBox	QDialog	menus.h	menus.cpp

Dialog box “Select model”.

name	base class	header file	code file
GeographyDialogBox	QDialog	menus.h	menus.cpp

Dialog box “Geography and migration”.

name	base class	header file	code file
LociDialogBox	QDialog	menus.h	menus.cpp

Dialog box “Loci and mutation”.

name	base class	header file	code file
FitnessDialogBox	QDialog	menus.h	menus.cpp

Dialog box “Individual and group fitness”.

name	base class	header file	code file
RunControlDialogBox	QDialog	menus.h	menus.cpp

Dialog box “Run control”.

name	base class	header file	code file
DataOutputDialogBox	QDialog	menus.h	menus.cpp

Dialog box “Data file output”.

name	base class	header file	code file
QAction	QObject	<QAction>	

Qt. Handles menu and pushbutton events.

name	base class	header file	code file
QMouseEvent	QSinglePointEvent	<QMouseEvent>	

Qt. Mouse click.

name	base class	header file	code file
AltruistView	QGraphicsView	graphics.h	graphics.cpp

Drawing of graphs, maps, etc.

name	base class	header file	code file
QGraphicsScene	QObject	<QGraphicsScene>	

A canvas for drawing geometric figures.

name	base class	header file
QGraphicsRectItem	QAbstractGraphicsShapeItem	<QGraphicsRectItem>

Draws a square.

name	base class	header file
QGraphicsTextItem	QGraphicsObject	<QGraphicsTextItem>

Draws text on a graphics canvas.

name	base class	header file
QGraphicsPolygonItem	QAbstractGraphicsShapeItem	<QGraphicsPolygonItem>

Draws a polygon.

name	base class	header file
QPointF		<QPointF>

Point on canvas. Point in polygon.

name	base class	header file
QPolygonF	QList	<QPolygonF>

List of points in polygon.

name	base class	header file	code file
QString		<QString>	

Qt. Dynamic text string.

name	base class	header file	code file
QThread	QObject	<QThread>	

Qt thread handler.

name	base class	header file	code file
QFile	QFileDevice	<QFile>	

Qt. Reading and writing files.

name	base class	header file	code file
QElapsedTimer		<QElapsedTimer>	

Measure elapsed time.

Version history

Version 3.003, 2025-01-13

Migrant number distribution changed from Poisson to binomial in territoriality models.

Version 3.002, 2024-10-12

Minor improvements. Some variable names changed.

Version 3.001, 2024-06-25

Only minor improvements

Version 3.00, 2024-01-01

The development of this program has been resumed after a long pause. The old program versions cannot run on current computers. A complete redesign of the program code was necessary. This version is using a 64-bit Windows platform with Qt graphics framework. It is designed to be portable to other platforms.

The random number generator is updated with improved random variate generation, Wallenius' and Fisher's noncentral hypergeometric distributions added, etc.

Regality model added. Improved graphics interface.

Version 2.04, 2001-01-06

Minor bug fix.

Version 2.03, 2000-04-17

Minor improvements.

Version 2.02, 2000-02-19

Group territoriality model improved.

Version 2.01, 1999-12-19

Haystack model and group territoriality models added. Source code published for the first time.

Version 2.00, 1997

Changed to 32-bit Windows 95 platform. Improved random number generator.

Version 1.00, 1995

Changed to 16-bit Windows 3.1 platform. Epistasis model added. Graphical user interface, using the now obsolete Borland object windows library. Each model is in a separate DLL.

Unnumbered versions, 1990-1993

Unpublished versions. 16-bit DOS 3 operating system. Island model, endogamy, and conformity.

The history of this program reflects an impressive development in computer technology. The execution speed has been improved by approximately five orders of magnitude since the first version in 1990. Versions 1 and 2 had critical parts coded in assembly language for the sake of performance. Fortunately, this is no longer necessary. Operating systems have also been improved. Unfortunately, the old versions cannot run in current operating systems, and neither can the compilers and graphics framework that were used for building them.

Open-source license